1. What are the key lessons you learned about web application security, and how do they relate to the CIA Triad?

The CIA Triad represents Confidentiality, Integrity, and Availability, the three core principles of information security. In the context of web application security, these principles guide the manner in which we protect data and services.

**Confidentiality** means preventing the unauthorized access of sensitive data. Web applications routinely deal with sensitive information (in the form of login credentials, personal information, payment-related information, etc.) hence the need for the use of data encryption (e.g. SSL/TLS), secure authentication modes (such as multi-factor authentication), and access controls.

**Integrity** defines the property of data not being altered or destroyed in an unauthorized way when stored or transmitted. Security of a web application is very essential in order to uphold the integrity of the data against unauthorized changes- for example, usage of SQL injection attacks to change content, or cross-site scripting (XSS) attacks to inject malicious scripts that change content. Input validation and proper sanitization can help effectively maintain integrity.

**Availability** signifies that the application and its data must be available and functioning when they are supposed to be. Denial-of-service (DoS) attacks, server outages, or overloads of malicious traffic should not interfere with web applications. Load balancing, redundancy, and protection against distributed denial-of-service (DDoS) attacks are important techniques for ensuring availability.

2. How do vulnerabilities and exploits affect web applications, and how can you defend against these attacks?

Weaknesses in a system, which attackers may exploit to gain unauthorized access or to cause some kind of damage, are called vulnerabilities. An exploit is a method or tool that an attacker uses in order to take advantage of a vulnerability.

Types of Vulnerabilities :

**SQL Injection:** An attacker can manipulate the SQL statements to execute arbitrary commands on the database.

**Cross-Site Scripting:** A malicious script is injected into a web page that is viewed by others.

**Cross-Site Request Forgery**: An attack where a user is tricked into performing an action that he does not want to do.

**Broken Authentication:** Weak authentication can be exploited by an attacker to impersonate a legitimate user.

**Insecure Deserialization:** Evoking arbitrary code via exploiting the inadequacy of the deserialization process.

**Defense Strategies:**

**Validate and sanitize input:** Make sure that any input coming into the system is safe before any backend interaction (i.e., using parameterized queries to avoid SQL injection attacks).

**Good practices for Authentication:** With Implementing Multi-factor Authentication (MFA), and Secure Password Storage (bcrypt) - Enforce Session Management Policies.

**Security Headers:** Utilize Security HTTP Headers (Content Security Policy - CSP) to prevent XSS and other injection attacks.

**Least Privilege Access:** Allow users to have the least amount of access necessary and implement Role-Based Access Control (RBAC).

**Regular Patching:** Keep on updating security software applications regularly in order to fix known vulnerabilities.

3. What role do different layers (client, server, database, etc.) play in web security, and what specific threats exist at each layer?

Numerous layers are implicated in web security, and every layer opens up its own vulnerabilities.

**Client Layer (Browser):**

**Threats:** XSS, Clickjacking, Session Hijacking, Defense: Content Security Policy (CSP) to restrict script execution, cookies configured to be secure (HttpOnly, SameSite), settings managed by the browser (disable third-party cookies).

**Server Layer:**

**Threats:** SQL injection, Remote Code Execution, and Unauthorized Access; Defense: Web Application Firewalls (WAFs), Input Validation, and Server Configuration (avoid default credentials, keep software current).

**Application Layer:**

**Threats:** Logic flaws, insecure deserialization, and authorization problems; Defenses: Secure code, proper role-based access controls, and proper testing of all logic flows.

**Database Layer:**

**Threats:** SQL injection, data leakage, and privilege escalation; Defense: Parameterized queries, data encryption, access controls for databases, and activity monitoring of databases.

**Network Layer:**

**Threats:** Man-in-the-middle attacks and DDoS attacks; Defense: HTTPS (TLS Encryption), VPNs, and DDoS protection services.

Each layer must be properly secured as an attacker will find and exploit weaknesses in one layer to gain access to others.

4. Discuss how web application security can fail in terms of configuration, policy, or assumptions. Provide an example you’ve learned about.

For a variety of technical reasons, web security failures are a common phenomenon owing to incorrect decisions concerning server or application configurations, weak security policies, or the assumption held by their designers that things are as expected. This brief is about common assumptions:

**Misconfiguration:** Considered an offending member of the weak controls behind many vulnerabilities found on the web. Examples are:

Default credentials left on web servers or databases (i.e., admin/admin).

Exposing sensitive information via public API endpoints or verbose error messages.

Policy Issues: Weak information security policies may cause:

Poor password policy (e.g., weak passwords allowed).

Failure to achieve either proper authorization or access control (e.g., unauthorized users accessing administration functions).

**Assumptions**: Security assumptions can foster vulnerabilities if they are not properly challenged. Some examples are:

Assuming all user inputs are safe implies no validation or sanitization may lead to possible XSS or SQL injection.

Having the assumption of a user being logged in from a trusted network leads to possibly inadequate session security.

**Example:** The Equifax incident involved, among other things, a failure to patch the known vulnerability of Apache Struts, which was exploited through weak patch management and configuration methods. Although the vulnerability had been known for months, the patch had not been installed, culminating in a massive data breach.

5. How do you think about risk and impact when evaluating web application security?

Risk-the likelihood of an attack-and impact-the results, should the attack succeed-are both valid considerations in rating the security of a web app. This becomes a basis by which security resources may be described. The rating process might look something like this:

**Risk Assessment:** Determining threats that could likely lead to the attack, such as SQL injection attacks or cross-site scripting (XSS) attacks; assessing how likely these are to happen; and evaluating how well the system is protected.

**Impact Assessment:** Considering the direct damages to the company as a result of this breach. For example:

In the case of a database being compromised-what does this mean for customer data, compliance (GDPR), reputation, etc.

In the case that the service is down, what is the effect on business operations and the customer?

Prioritize those issues that are both high-risk and high-impact. For example, vulnerability in authentication such as weak password policy may be categorized as a high risk, but a low impact breach if the application does not store sensitive information.

6. What prevention strategies have you found most effective.

As preventing security incidents is always better than trying to mitigate their consequences, some of the effective prevention strategies are:

**Secure Development:** Secure coding efforts should focus on input validation, output encoding, and the principle of least privilege.

**Automated Security Testing:** Early detection is the key to preventing a security incident. Vulnerability scanning, static analysis, and dynamic analysis should be done through automated tools.

**Security by Design**: Start the security of the application with design rather than implementing it afterward. Some of the security features that need to be in development are: Authentication and Authorization, and Input Validation.

**Patch Management**: Managing the patches means keeping the software along with environment and libraries regularly updated so that they reduce the scope of known vulnerabilities.

**Incident Response Planning:** Creation and exercising of incident response plans will assist in the prompt mitigation and recovery while under attack.

To keep web applications secure, there needs to be set in place a comprehensive process that binds secure coding practices, pre-emptive vulnerability testing, and timely patching.