**Experiment 5**

**Aim** - Study of OWASP vulnerabilities.

**Theory**- The **study of OWASP vulnerabilities** focuses on understanding the most common security risks found in web applications, as defined by the **Open Web Application Security Project (OWASP)**. OWASP publishes a widely recognized list called the **OWASP Top 10**, which highlights the most critical security vulnerabilities affecting web applications.

### **What is OWASP?**

The **Open Web Application Security Project (OWASP)** is a non-profit organization that provides free, open-source resources focused on improving the security of software. The **OWASP Top 10** is its flagship project and serves as a benchmark for web application security, identifying the most prevalent security risks based on data collected from security experts worldwide.

### **OWASP Top 10 Vulnerabilities**

#### **1. Broken Access Control**

* **Description**: Access control vulnerabilities occur when users are allowed to perform actions or access resources that they shouldn't. For example, a user with a low level of privileges could access administrative functions.
* **Examples**:
  + Modifying URLs or HTTP requests to access unauthorized data.
  + Privilege escalation, where a user gains unauthorized access to higher-level functions.
* **Mitigation**:
  + Implement proper role-based access control (RBAC).
  + Ensure access control checks are done on the server-side and not just client-side.

#### **2. Cryptographic Failures (Previously “Sensitive Data Exposure”)**

* **Description**: Cryptographic failures happen when sensitive data (like passwords, credit card numbers, or health records) is not properly protected in storage or transit.
* **Examples**:
  + Lack of encryption for sensitive data.
  + Use of weak encryption algorithms or improper key management.
* **Mitigation**:
  + Use strong encryption standards (e.g., AES-256) for sensitive data storage and transmission.
  + Properly manage encryption keys and avoid hard-coding them into applications.

#### **3. Injection (SQL, NoSQL, Command Injection)**

* **Description**: Injection attacks occur when an attacker can insert malicious code (such as SQL commands) into an application, tricking the application into executing unintended commands.
* **Examples**:
  + SQL injection, where an attacker can modify or retrieve sensitive information from a database.
  + Command injection, where an attacker executes system-level commands on the server.
* **Mitigation**:
  + Use prepared statements and parameterized queries.
  + Validate and sanitize all user inputs.

#### **4. Insecure Design**

* **Description**: Insecure design refers to security flaws at the architectural level, where security measures were not considered during the initial design of the application.
* **Examples**:
  + Lack of security controls, such as insufficient logging or no rate limiting on APIs.
  + Improperly designed data flows that expose sensitive information.
* **Mitigation**:
  + Incorporate security best practices during the design phase, including threat modeling.
  + Regularly review and update the application's design to meet security requirements.

#### **5. Security Misconfiguration**

* **Description**: This occurs when systems or applications are not securely configured. Default settings that are not secure, incomplete configurations, or improper access permissions can lead to vulnerabilities.
* **Examples**:
  + Leaving default usernames and passwords unchanged.
  + Leaving error messages exposed that reveal sensitive details about the application's architecture.
* **Mitigation**:
  + Ensure secure configurations for servers, databases, and applications.
  + Regularly update software and patch known vulnerabilities.

#### **6. Vulnerable and Outdated Components**

* **Description**: Using outdated software components with known vulnerabilities exposes an application to attacks. This includes frameworks, libraries, and modules that are no longer supported.
* **Examples**:
  + Using a vulnerable version of a JavaScript library (e.g., jQuery or React).
  + Running outdated versions of backend technologies like Apache or Nginx.
* **Mitigation**:
  + Regularly update all software components and libraries.
  + Use automated tools to track and alert about outdated dependencies.

#### **7. Identification and Authentication Failures**

* **Description**: These vulnerabilities arise when the authentication process (verifying user identity) or session management is weak, allowing attackers to assume the identity of other users.
* **Examples**:
  + Weak password policies or storing passwords in plain text.
  + Failure to securely manage sessions, allowing session hijacking.
* **Mitigation**:
  + Implement multi-factor authentication (MFA).
  + Use secure session management with features like session expiration and invalidation.

#### **8. Software and Data Integrity Failures**

* **Description**: This risk occurs when software or data is not properly validated for integrity, leaving applications vulnerable to supply chain attacks or unauthorized data modifications.
* **Examples**:
  + Using third-party libraries that have been compromised.
  + Insufficiently validating the integrity of updates and code changes.
* **Mitigation**:
  + Use digital signatures and checksums to verify the integrity of software and updates.
  + Implement secure coding practices to ensure that data integrity checks are built into applications.

#### **9. Security Logging and Monitoring Failures**

* **Description**: Lack of effective logging and monitoring makes it difficult to detect and respond to security incidents in a timely manner.
* **Examples**:
  + Not logging critical security-related events (e.g., failed login attempts).
  + Insufficient monitoring that allows attacks like brute force or account takeovers to go unnoticed.
* **Mitigation**:
  + Ensure proper logging of security-related events, such as authentication failures and access control checks.
  + Regularly review and monitor logs for unusual or suspicious activity.

#### **10. Server-Side Request Forgery (SSRF)**

* **Description**: SSRF vulnerabilities occur when an application fetches resources from user-specified URLs without proper validation, allowing attackers to access internal resources.
* **Examples**:
  + An attacker provides a URL to an internal service that shouldn’t be accessible, exploiting it to access internal systems or services.
* **Mitigation**:
  + Validate all incoming URLs and ensure they are within an allowlist.
  + Use network segmentation to limit access to sensitive internal services.

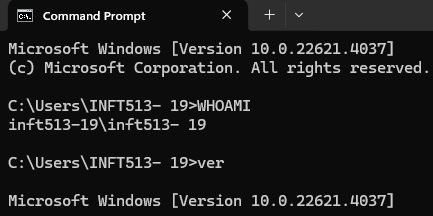
### **Importance of Studying OWASP Vulnerabilities:**

1. **Understanding Common Threats**: By studying OWASP vulnerabilities, developers and security professionals can better understand the most common security risks faced by web applications, and how these risks are exploited by attackers.
2. **Prevention and Mitigation**: Familiarity with OWASP vulnerabilities helps in adopting best practices to prevent security flaws during the development cycle, such as input validation, proper authentication, and access control.
3. **Compliance and Standards**: Many industries require applications to follow certain security standards (e.g., PCI-DSS, HIPAA). The OWASP Top 10 serves as a baseline for building secure applications and meeting regulatory requirements.
4. **Security Awareness**: Developers can incorporate security practices into their development workflow, moving from reactive to proactive security management.

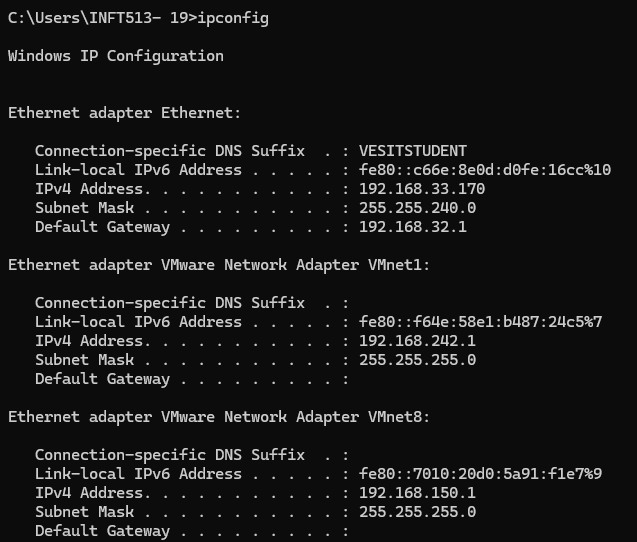
Below is a summary of some commands that are useful on Linux and Windows

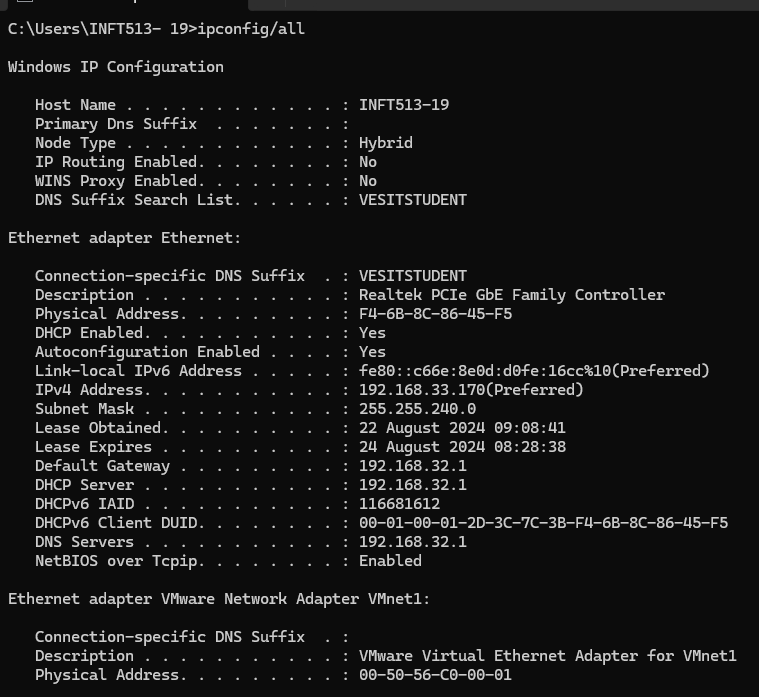
platforms:

1. **Name of current user-:** **whoami**
2. **Operating system-: ver**

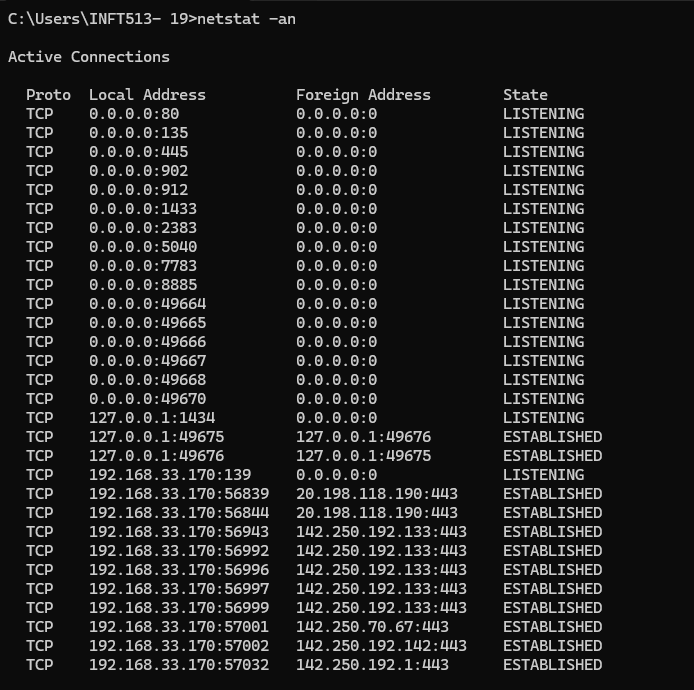
****

1. **Network configuration-: ipconfig /all**

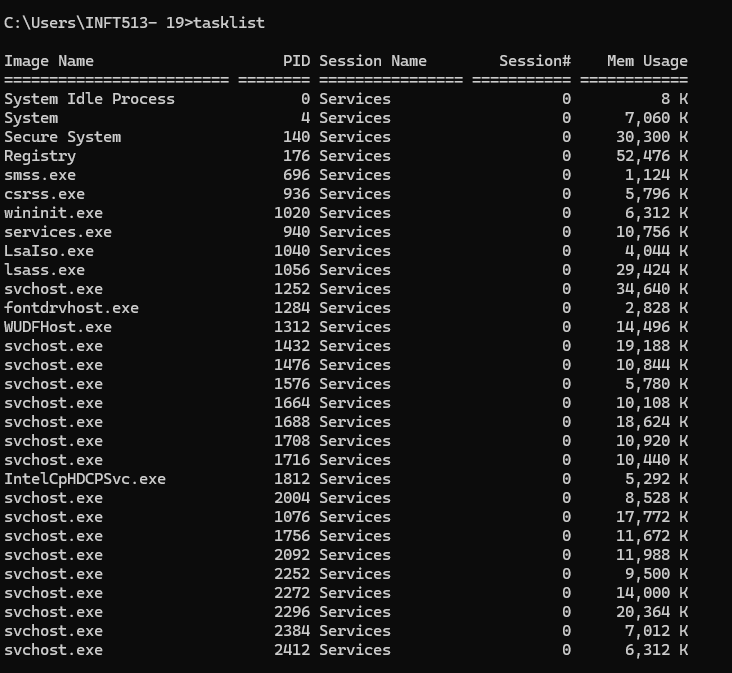
****

****

1. **Network connections-: netstat -an**

****

1. **Running processes-: tasklist**

****

**Conclusion**:In conclusion, the study of **OWASP vulnerabilities** is essential for developers and security professionals who aim to build and maintain secure web applications. By understanding and addressing the vulnerabilities outlined in the **OWASP Top 10**, teams can reduce the likelihood of common attacks and build more robust, secure systems.