Unit-1.0: Fundamentals of Web Application Security (7 Hours)

1. Introduction to Web Application Security

Web Application Security refers to the set of principles, techniques, and practices used to protect web applications from cyber threats, unauthorized access, data leakage, and misuse. Because web applications are always connected to the internet, they are highly vulnerable to attacks such as SQL injection, XSS, CSRF, and brute force attacks.

Web security ensures:

- Confidentiality (data is protected)
- Integrity (data is not altered)
- Availability (services work properly)
- Authentication (verify identity)
- Authorization (access control)

---

2. History of Software Security

Software security evolved in response to increasing cyber threats:

Early Stage (1960s–1980s)

- Computers were used only by professionals.
- Security focused on physical access to systems.
- No major concept of "software security."

1990s

- Internet became common; web applications began to emerge.
- First major cyberattacks appeared:
    - Buffer overflow attacks
    - Password cracking
    - Basic malware
- Developers didn't consider security during development.

2000s

- Rise of e-commerce and online transactions.
- Web application attacks increased drastically.
- OWASP (Open Web Application Security Project) was formed (2001).

- SQL Injection and XSS became common attack methods.

2010–Present

- Cloud computing, mobile apps, microservices, APIs.
- Attackers became more advanced (phishing, ransomware, DDoS).
- Secure Software Development Life Cycle (SSDLC) practices introduced.
- Security automation, penetration testing, and DevSecOps became standard.

Conclusion:
Modern software security focuses on "secure-by-design," where security is included in every stage of development.

---

3. Recognizing Web Application Security Threats

Web apps face different types of vulnerabilities. Major threats include:

1. Injection Attacks

- SQL Injection, Command Injection.
- Attackers insert malicious input.
- Can access, delete, or modify entire database.

2. Cross-Site Scripting (XSS)

- Inject malicious scripts in webpages.
- Steals cookies, session tokens.

3. Cross-Site Request Forgery (CSRF)

- Forces users to perform actions without consent.

4. Broken Authentication

- Weak login process.
- Attackers impersonate other users.

5. Broken Access Control

- Users can access unauthorized resources.

6. Security Misconfigurations

- Default passwords, exposed admin panels.

7. Sensitive Data Exposure

- Unencrypted passwords, personal data leakage.

8. Denial of Service (DoS/DDoS)

- Attackers overload server to shut it down.

9. Vulnerable APIs

- Improper validation allows attackers to misuse APIs.

Recognizing these threats is the first step in securing the application.

---

4. Web Application Security

Web application security involves protection measures like:

1. Secure Coding Practices

- Input validation
- Parameterized queries
- Avoiding hard-coded passwords

2. Security Testing

- Vulnerability scanning
- Penetration testing
- Code review

3. Deployment Security

- HTTPS configuration
- Server hardening
- Firewall and IDS/IPS

4. Monitoring & Logging

- Track suspicious activities
- Detect brute force attempts
- Monitor server logs

5. OWASP Top 10

A standard guideline for most common vulnerabilities in web applications.

---

5. Authentication and Authorization

⭐ Authentication

Authentication means verifying the identity of a user.

Common Authentication Methods

1. **Password-Based Authentication**
2. **Multi-Factor Authentication (MFA)**
3. **Biometric Authentication**
4. **Token-Based Authentication**
5. **OAuth / OpenID Connect**
6. **Certificate-Based Authentication**

Common Authentication Attacks

- Brute force
- Credential stuffing
- Password spraying
- Phishing

⭐ Authorization

Authorization determines **what resources an authenticated user can access.**

Example:

- User logs in → Authentication
- User can only see their data → Authorization

Authorization Models

1. **Role-Based Access Control (RBAC)**
2. **Attribute-Based Access Control (ABAC)**
3. **Access Control Lists (ACL)**

Common Authorization Issues

- Privilege escalation
- Insecure direct object reference (IDOR)

---

6. Secure Socket Layer (SSL)

SSL (Secure Socket Layer) is a protocol used to secure communication between a client (browser) and a server.

Functions of SSL

- Encrypts data during transmission
- Protects data from eavesdropping
- Verifies server identity

How SSL Works

1. Client sends request → "Hello"
2. Server sends digital certificate
3. Client verifies certificate
4. A secure session is established
5. All data is encrypted

Limitations

- Old versions are vulnerable
- Replaced by TLS

---

7. Transport Layer Security (TLS)

TLS is the improved and more secure version of SSL.

Features of TLS

- Strong encryption
- Integrity checking
- Authentication
- Protection against replay attacks

TLS Versions

- TLS 1.0 (1999)
- TLS 1.1 (2006)
- TLS 1.2 (2008 – widely used)

- TLS 1.3 (2018 – fastest, most secure)

Importance of TLS

- Used in all secure websites (HTTPS)
- Protects online banking, shopping, login forms

---

8. Session Management

A session is a temporary interaction between a user and a web application.

Key Components

1. **Session ID**
2. **Cookies**
3. **Session timeout**
4. **Session storage on server**

Common Session Attacks

1. Session Hijacking
2. Session Fixation
3. Cookie Tampering
4. Man-in-the-Middle (MITM) attack

Best Practices

- Regenerate session ID after login
- Use Secure and HttpOnly flags in cookies
- Short session timeout
- Use HTTPS everywhere

---

9. Input Validation

Input validation ensures data entered by users is safe.

Why Input Validation is Important?

- Prevents SQL Injection
- Prevents XSS

- Prevents buffer overflows
- Ensures data integrity

Types of Input Validation

1. **Client-side validation**
    - Done using JavaScript
    - Quick but not secure
2. **Server-side validation**
    - More secure
    - Always must be done

Best Practices

- Never trust user input
- Use allowlist validation
- Use prepared statements
- Validate input length, type, format

Summary of Unit–1

Topic	Summary
History of Software Security	Evolution from basic systems to modern advanced security practices
Security Threats	Injection, XSS, CSRF, Broken Auth, DoS, etc.
Authentication	Verify identity
Authorization	Control access
SSL	Older security protocol
TLS	Modern secure protocol
Session Management	Secure sessions using cookies
Input Validation	Prevents user-input-based attacks

Unit-2.0: Secure Development and Deployment (7 Hours)

Web applications must be designed, developed, tested, deployed, and maintained using secure practices. This unit focuses on modern frameworks and processes that ensure security is integrated throughout the Software Development Life Cycle (SDLC).

---

1. Web Application Security in Development and Deployment

Developing secure web applications involves combining **secure coding**, **secure design**, and **secure deployment** techniques.

Key Principles

1. **Security by Design**
    - Security requirements defined at the beginning of the project.
2. **Defense in Depth**
    - Multiple layers of security (firewalls + authentication + encryption + validation).
3. **Least Privilege**
    - Give only required access to users and system components.
4. **Secure Deployment**
    - Secure hosting
    - Hardened servers
    - Proper configuration
    - Regular updates and patching
5. **Continuous Security Monitoring**
    - Detect suspicious activities
    - Log all security events

---

2. Security Testing

Security testing ensures that vulnerabilities in a web application are identified before deployment.

Objectives of Security Testing

- Identify security weaknesses
- Prevent unauthorized access

- Ensure data confidentiality and integrity
- Validate authentication and authorization
- Ensure compliance with standards (OWASP, NIST, ISO 27001)

Types of Security Testing

1. Vulnerability Assessment

- Automated scanning to detect common weaknesses.
- Tools: Nessus, OpenVAS, Nikto.

2. Penetration Testing

- Ethical hacking to exploit vulnerabilities like an attacker.
- Tools: Burp Suite, Metasploit, Kali Linux.

3. Static Application Security Testing (SAST)

- Source code analysis without running the application.
- Checks for insecure coding practices.
- Tools: SonarQube, Checkmarx.

4. Dynamic Application Security Testing (DAST)

- Tests the running application from outside like a black-box test.
- Tools: OWASP ZAP, Burp Scanner.

5. Interactive Application Security Testing (IAST)

- Combines SAST + DAST during runtime.

6. Security Regression Testing

- After bug fixes, re-test security functions.

7. API Security Testing

- Verifies API authentication, rate limiting, authorization.

---

3. Security Incident Response Planning

Security incidents are events that threaten the security of an application or system. Examples: data breach, malware attack, unauthorized login, server compromise.

A **Security Incident Response Plan (SIRP)** ensures that an organization can quickly identify, analyze, and recover from such incidents.

Phases of Incident Response

1. Preparation

- Create an incident response team
- Tools for monitoring and logging
- Backup system
- Security training

2. Identification

- Detect and confirm the incident
- Analyze logs, alerts, error messages
- Determine the severity

3. Containment

- Short-term containment: Stop the attack temporarily
- Long-term containment: Fix vulnerabilities and isolate affected systems

4. Eradication

- Remove root cause: malware, vulnerabilities, misconfigurations
- Apply patches
- Harden systems

5. Recovery

- Restore normal operations
- Monitor system for further attacks
- Validate system stability

6. Lessons Learned

- Document incident
- Improve the security process
- Update policies and security controls

Importance:
Reduces damage, prevents future attacks, protects data, and ensures business continuity.

---

4. The Microsoft Security Development Lifecycle (SDL)

The **Microsoft SDL** is a security assurance process designed to integrate security at every stage of software development.

Goals of SDL

- Reduce vulnerabilities
- Improve software reliability
- Ensure compliance with security standards

Phases of Microsoft SDL

1. Training

- Developers receive security training
- Topics: secure coding, common vulnerabilities

2. Requirements

- Establish security/privacy requirements
- Risk assessment
- Define quality gates

3. Design

- Threat modeling
- Attack surface analysis
- Architecture review

4. Implementation

- Secure coding guidelines
- Use approved libraries
- Remove unsafe functions

5. Verification

- Code analysis (SAST)
- Penetration testing
- Fuzz testing
- Security testing tools

6. Release

- Final security review

- Incident response plan ready
- Sign-off document

7. Response

- Security monitoring
- Updates and patches
- Post-release incident handling

Benefits

- Reduces cost of fixing bugs (cheaper during development)
- Improves customer trust
- Standardized and repeatable security process

---

5. OWASP Comprehensive Lightweight Application Security Process (CLASP)

CLASP is a security process framework created by OWASP to provide a structured method for building secure software.

Why CLASP?

- Simple to adopt
- Works for small and large teams
- Focuses on practical actionable steps

CLASP Key Components

1. Security Requirements and Policies

Defines what security controls the application must follow.

2. Secure Design

- Threat modeling
- Security architecture review

3. Secure Coding

- Coding guidelines
- Prevent insecure features
- Reduce attack surface

4. Security Testing

- Automated tools
- Manual review
- Penetration testing

5. Vulnerability Management

- Track vulnerabilities
- Assign priority
- Apply fixes

CLASP Activities (Main Activities)

1. Identifying security roles and responsibilities
2. Performing risk analysis
3. Defining secure coding practices
4. Conducting design review
5. Performing code review
6. Conducting security testing

CLASP is highly modular, meaning organizations can adopt only the parts they need.

---

6. The Software Assurance Maturity Model (SAMM)

SAMM is an open-source framework developed by **OWASP** for measuring and improving an organization's software security posture.

Purpose of SAMM

- Assess current security practices
- Identify weaknesses
- Provide maturity levels for improvement
- Applicable to all types of development methodologies (Agile, DevOps, Waterfall)

SAMM Structure

SAMM consists of **4 Business Functions**, each containing **3 Security Practices**:

Business Function	Security Practices
1. Governance	Strategy & Metrics, Policy & Compliance, Education & Guidance
2. Design	Threat Assessment, Security Requirements, Secure Architecture
3. Implementation	Secure Build, Secure Deployment, Environment Hardening
4. Verification	Architecture Assessment, Requirements Testing, Security Testing

---

SAMM Maturity Levels

Each practice has **3 maturity levels**:

Level 1: Basic

- Ad-hoc, limited processes
- Some security awareness

Level 2: Intermediate

- Defined, repeatable processes
- Regular security testing

Level 3: Advanced

- Fully integrated security programs
- Continuous improvement
- Automated and optimized processes

Benefits of SAMM

- Helps organizations understand their current maturity
- Provides roadmap for improvement
- Supports compliance with standards (ISO, OWASP, NIST)
- Works with Agile, DevOps, CI/CD environments

Summary of Unit 2

Topic	Summary
Security Testing	SAST, DAST, Pen-testing, API testing, vulnerability assessment
Incident Response	Preparation → Identification → Containment → Eradication → Recovery → Lessons Learned
Microsoft SDL	End-to-end security in SDLC: requirements, design, implementation, testing, release, response
OWASP CLASP	Lightweight, practical security process focusing on roles, design, coding, testing
OWASP SAMM	Maturity model for assessing and improving software security practices

Unit-3.0: Secure API Development (7 Hours)

APIs (Application Programming Interfaces) are the backbone of modern web and mobile applications. They enable communication between client–server, microservices, and cloud services. Therefore, securing APIs is critical to prevent data breaches, unauthorized access, and misuse.

This unit explains various techniques and standards used to secure APIs.

---

1. API Security

API Security refers to protecting APIs from attacks and unauthorized usage. APIs often expose sensitive data and functionalities, making them major targets for attackers.

Common API Threats

- Unauthorized access
- Broken authentication
- Injection attacks (SQL, NoSQL, Command Injection)
- Data exposure
- Replay attacks
- DDoS rate abuse
- Man-in-the-middle attacks
- Insecure API tokens or keys

Principles of API Security

1. **Least privilege**
2. **Zero trust security model**
3. **Strong authentication and authorization**
4. **Input validation**
5. **Encryption everywhere (HTTPS/TLS)**
6. **Centralized logging and monitoring**

---

2. Session Cookies

Traditional websites use **session cookies** to identify authenticated users.

How Session Cookies Work

1. User logs in
2. Server generates a **session ID**
3. Session ID stored in **browser cookie**
4. Browser sends cookie with every request

Security Risks

- Session hijacking
- Session fixation
- Cookie theft (via XSS)

Security Controls

- Use **Secure flag** → cookie sent only via HTTPS
- Use **HttpOnly flag** → JavaScript cannot access cookie
- Use **SameSite flag** → prevents CSRF
- Regenerate session ID after login
- Set short expiration time

---

3. Token-Based Authentication

Token-based authentication replaces session cookies and is ideal for APIs, mobile apps, and microservices.

How It Works

1. User logs in
2. Server returns a token (e.g., JWT – JSON Web Token)
3. Client stores token (localStorage or memory)
4. Client sends token in **Authorization header**:
5. Authorization: Bearer <token>

Advantages

- Stateless (no server-side sessions)
- Scalable
- Works across multiple platforms
- More secure for APIs

Token Types

- JWT (JSON Web Token)
- OAuth Access Tokens
- Refresh Tokens
- API Tokens

Token Security Practices

- Short expiration
- Rotate and refresh tokens
- Signature verification
- Do not store JWT in localStorage (use HttpOnly cookie or memory)
- Validate token on every request

---

4. Securing "Natter APIs": Addressing Threats with Security Controls

Natter APIs refers to general web APIs that must be secured using standard techniques.

Common Threats and Controls

Threat	Security Control
Injection	Validate input, sanitize data, parameterized queries
Broken Auth	Strong authentication, MFA, secure tokens
Sensitive Data Exposure	Use TLS, encrypt data at rest
Replay Attacks	Use timestamps, nonces
Rate Abuse	Rate limiting, throttling
MITM Attacks	HTTPS only
IDOR	Proper authorization checks
CSRF	SameSite cookies, anti-CSRF tokens

---

5. Rate Limiting for Availability

Rate limiting protects API from DDoS, brute-force, and abuse.

What is Rate Limiting?

Restricting the number of API requests client can make in a certain time.

Techniques

1. **Fixed Window Counter**
2. **Sliding Window Counter**
3. **Token Bucket Algorithm**
4. **Leaky Bucket Algorithm**

Benefits

- Prevents server overload
- Reduces brute force passwords attempts
- Improves availability
- Ensures fair usage for all users

---

6. Encryption

Encryption ensures confidentiality and integrity of sensitive data.

Types of Encryption

1. **In-transit encryption**
    - Use **TLS/HTTPS**
    - Prevents MITM attacks
2. **At-rest encryption**
    - Encrypt API databases, tokens, API secrets

Best Practices

- Use TLS 1.2+ / TLS 1.3
- Use strong ciphers (AES-256, RSA-2048+)
- Hash passwords using bcrypt/Argon2
- Never store sensitive data in plain text

---

7. Audit Logging

Audit logs track all activities happening inside API systems.

Log What?

- Authentication attempts
- Failed login attempts
- API key usage
- Data access logs
- Errors and anomalies
- Admin actions

Importance

- Helps during incident response
- Helps detect brute-force attacks
- Needed for compliance (GDPR, ISO 27001)

Best Practices

- Use centralized logging (ELK, Splunk)
- Log timestamp, IP, user ID
- Protect logs from tampering

---

8. Securing Service-to-Service APIs

Microservices communicate with each other using internal APIs.

Key Methods

A. API Keys

Simple identifiers for authenticating service-to-service calls.

Features

- Attached in header or query string
- Easy to use
- Not user-specific

Security Guidelines

- Store keys securely (environment variables, vaults)
- Rotate keys regularly
- Never hard-code in source code
- Use separate key per service

B. OAuth2

OAuth2 is the industry standard framework for secure access delegation.

Why OAuth2?

- More secure than API Keys
- Provides authorization + token management
- Used by Google, Facebook, GitHub, Microsoft

OAuth2 Flow Examples

1. **Client Credentials Flow** (service-to-service)
2. **Authorization Code Flow** (user login flow)
3. **Implicit Flow** (legacy)
4. **Device Flow**

Tokens in OAuth2

- Access Token
- Refresh Token

Security Benefits

- Granular access control
- Expiring tokens
- Secure delegation

---

9. Securing Microservice APIs

Microservices architecture requires additional security controls.

A. Service Mesh

A **service mesh** (e.g., Istio, Linkerd) provides secure communication between microservices without modifying code.

Functions of Service Mesh

- Automatic TLS encryption between services
- Service identity and authentication
- Traffic control
- Observability and logging
- Policy enforcement

Advantage:
Security happens at the infrastructure layer, not in application code.

B. Locking Down Network Connections

Microservices must use **zero trust networking**.

Techniques

1. **Network segmentation**
2. **Private subnets**
3. **Firewall rules**
4. **Security groups**
5. **Mutual TLS (mTLS)**

mTLS Features

- Both client and server authenticate each other
- Protects internal service calls

C. Securing Incoming Requests

Incoming API requests must be validated and filtered.

Methods

1. **Input Validation**
2. **Request Authentication**
3. **Authorization checks**

4. **Signature verification** (HMAC signatures)
5. **Anti-bot protections**
6. **IP filtering / allowlist**
7. **Web Application Firewall (WAF)**

Summary of Unit-3

Topic	Summary
API Security	Protect APIs from unauthorized access, injection, replay, and MITM attacks
Session Cookies	Use Secure, HttpOnly, SameSite flags
Token Authentication	Stateless authentication using tokens like JWT
Threat Controls	Input validation, rate limiting, encryption, authorization
Rate Limiting	Prevents DDoS and abuse
Encryption	HTTPS/TLS, AES, RSA
Audit Logging	Track events, detect anomalies
Service-to-Service Security	API Keys, OAuth2
Microservice API Security	Service Mesh, mTLS, network lockdown

Unit-4.0: Vulnerability Assessment and Penetration Testing (VAPT)

VAPT is a combined process used to evaluate the security posture of an organization. It identifies weaknesses (Vulnerability Assessment) and attempts to exploit them (Penetration Testing).

---

1. Vulnerability Assessment Lifecycle

Vulnerability Assessment (VA) is a structured process to identify, classify, and prioritize vulnerabilities in systems, networks, databases, and applications.

Lifecycle Steps

1. Planning

- Define scope of assessment
- Identify assets (servers, networks, applications)
- Determine testing method (automated/manual)
- Set rules of engagement

2. Vulnerability Discovery

- Perform scanning
- Identify outdated software, misconfigurations, missing patches
- Detect weak authentication, open ports, insecure services

Techniques used:

- Automated scanning
- Manual analysis
- Configuration review
- Asset discovery tools

3. Vulnerability Analysis

- Validate scan results
- Remove false positives
- Analyze risk level: High / Medium / Low
- Map vulnerabilities to CVE/CVSS score

4. Risk Assessment & Prioritization

Prioritize vulnerabilities based on:

- Severity (CVSS score)
- Exploitability
- Business impact
- Asset importance

5. Reporting

Prepare a structured report including:

- Summary of findings
- Vulnerability details
- Severity levels
- Potential business impact
- Recommended remediation steps

6. Remediation

- Apply patches
- Fix misconfigurations
- Update software
- Improve network policies

7. Re-Assessment / Verification

- Test again to ensure vulnerabilities are fixed
- Confirm remediation success
- Close the assessment cycle

---

2. Vulnerability Assessment Tools

These tools help detect vulnerabilities in different layers of IT infrastructure.

A. Cloud-Based Vulnerability Scanners

Used to scan cloud environments (AWS, Azure, GCP).

Examples

- Qualys Cloud Platform

- Nessus Cloud
- Rapid7 InsightVM

Features

- No installation needed
- Real-time scanning
- Cloud workload security
- Compliance monitoring
- Automated reporting

Use Cases

- Scan cloud servers
- Check storage bucket security
- Verify misconfigured IAM policies

B. Host-Based Vulnerability Scanners

Installed directly on a host (laptop, server, VM) to check system-level vulnerabilities.

Examples

- OSSEC
- Microsoft Baseline Security Analyzer (MBSA)
- Lynis

Detects

- Missing patches
- Weak passwords
- Open ports
- File integrity issues
- Malware traces

Best For

- Individual workstations
- Linux/Windows servers

- Endpoint security checks

C. Network-Based Vulnerability Scanners

These tools scan entire networks to detect vulnerabilities in routers, firewalls, switches, and servers.

Examples

- Nmap
- Nessus
- OpenVAS
- Qualys Network Scanner

Detects

- Open/unused ports
- Insecure network services
- OS fingerprinting
- Firewall misconfigurations

Use Cases

- Enterprise networks
- Data centers
- Internet-facing servers

D. Database-Based Vulnerability Scanners

Specialized tools for scanning databases.

Examples

- IBM Guardium
- Oracle Database Assessment Tool
- SQLMap (for injection testing)

Detects

- SQL injection vulnerabilities
- Weak database passwords

- Misconfigured privileges
- Unencrypted sensitive data
- Database patch issues

Used For

- MySQL, PostgreSQL, Oracle, SQL Server, MongoDB
- Securing sensitive customer and financial data

---

3. Types of Penetration Tests

Penetration Testing is a simulation of a real-world attack to exploit vulnerabilities and assess the security level.

A. External Penetration Testing

Testing from outside the network (public internet).

Objectives

- Identify public-facing vulnerabilities
- Attempt to access internal systems
- Test firewalls, servers, APIs, DNS

Targets

- Websites
- Web servers
- Email servers
- Exposed APIs
- Cloud services

Useful For

- Detecting open ports
- Testing firewall rules
- Finding misconfigured DNS
- Discovering externally exploitable vulnerabilities

B. Web Application Penetration Testing

Focuses on security of web apps.

Techniques

- SQL Injection
- Cross-Site Scripting (XSS)
- Cross-Site Request Forgery (CSRF)
- Broken authentication
- Sensitive data exposure
- Insecure API endpoints

Tools

- Burp Suite
- OWASP ZAP
- Nikto
- SQLMap

Outcome

- Secured forms, login pages, APIs
- Stronger authentication and session management

C. Internal Penetration Testing

Simulates an attack from inside the organization (trusted network).

Performed After

- Security breach
- Insider attack investigation

Checks

- Internal network segmentation
- Lateral movement threats
- Privilege escalation
- Weak internal passwords

- File sharing weaknesses

Tools

- Metasploit
- PowerShell Empire

D. SSID or Wireless Penetration Testing

Tests Wi-Fi and wireless network security.

Objectives

- Identify weak Wi-Fi encryption
- Exploit default or weak passwords
- Identify rogue access points
- Test wireless network isolation

Tools

- Aircrack-ng
- Wireshark
- Kismet

Threats Found

- WPA/WPA2 vulnerabilities
- Evil twin attacks
- Packet sniffing

E. Mobile Application Penetration Testing

Targets mobile apps on Android and iOS.

Checks

- Insecure data storage
- Weak encryption
- Insecure API communication
- Hardcoded keys in app

- Code tampering

Tools

- MobSF (Mobile Security Framework)
- Drozer
- Frida/Objection

Test Areas

1. Application permissions
2. API calls
3. Local data encryption
4. Jailbreak/root detection
5. Secure input and output

Summary of Unit-4

Topic	Summary
VA Lifecycle	Planning → Discovery → Analysis → Reporting → Remediation → Reassessment
Cloud Scanners	Scan cloud assets like AWS, Azure
Host Scanners	Scan OS-level vulnerabilities
Network Scanners	Scan entire networks
Database Scanners	Identify DB-related risks
External Pen Testing	Test external/public-facing assets
Web App Testing	Identify OWASP Top 10 issues
Internal Testing	Simulates insider attack
Wireless Testing	Check Wi-Fi encryption and access
Mobile Testing	Analyze Android/iOS security

Unit-5.0: Hacking Techniques and Tools (7 hrs)

1. Social Engineering

Social engineering is the technique of manipulating people into revealing confidential information.
Instead of attacking computers, attackers exploit human psychology.

Types of Social Engineering Attacks

1. **Phishing**
   Fake emails or websites trick users into entering login credentials or sensitive data.
2. **Spear Phishing**
   Targeted phishing aimed at specific individuals or organizations.
3. **Vishing**
   Voice phishing—calls pretending to be banks, IT support, etc.
4. **Smishing**
   Phishing using SMS messages.
5. **Pretexting**
   Attacker creates a false scenario to gain trust and ask for data (e.g., pretending to be HR).
6. **Baiting / USB drops**
   Attackers leave infected USB drives, hoping users will plug them in.
7. **Tailgating / Piggybacking**
   Gaining unauthorized physical access by following an authorized user.

Defense

- Employee security awareness training
- MFA (Multi-Factor Authentication)
- Email filtering systems
- Zero Trust security model

---

2. Injection Attacks

Injection occurs when untrusted input is sent to a server, causing it to interpret malicious commands.

Types

1. **SQL Injection**
   Attacker modifies SQL queries.
   Example: ' OR '1'='1
2. **Command Injection**
   Executes system commands on the server.
3. **LDAP Injection**
   Exploits directory queries like Active Directory.
4. **XML Injection (XXE)**
   Injecting malicious XML entities.

Consequences

- Database theft and corruption
- Bypassing authentication
- Remote code execution

Prevention

- Input validation (whitelisting)
- Prepared statements / parameterized queries
- Escaping special characters

---

3. Cross-Site Scripting (XSS)

XSS allows attackers to inject malicious JavaScript into web pages viewed by other users.

Types

1. **Stored XSS**
   Script is permanently stored on the server (e.g., comment section).
2. **Reflected XSS**
   Script is part of the request and reflected in server response.
3. **DOM-based XSS**
   Exploit occurs in the browser (client-side JavaScript).

Effects

- Session hijacking
- Cookie theft
- Redirecting users to malicious sites

- Defacing websites

Prevention

- Escape HTML output
- Content Security Policy (CSP)
- Sanitization libraries

---

4. Broken Authentication and Session Management

Weaknesses in handling user authentication and sessions allow attackers to impersonate legitimate users.

Examples

- Predictable session IDs
- Session ID exposed in URLs
- No session timeout
- Weak password policies
- Credential stuffing (reusing leaked passwords)

Attacks

- Session Hijacking
- Session Fixation
- Password brute-force attacks

Prevention

- Use secure session tokens
- Implement MFA
- Enforce strong passwords and lockout mechanisms
- HTTPS-only cookies

---

5. Cross-Site Request Forgery (CSRF)

CSRF forces a logged-in user to perform actions they did not intend by sending unauthorized requests from their browser.

Example

A banking site auto-submits a money transfer request without user consent.

Defense

- Anti-CSRF tokens
- SameSite cookies
- Confirm-action pages for sensitive tasks

---

6. Security Misconfiguration

Occurs when security settings are left incorrect or default.

Examples

- Default admin usernames/passwords
- Unnecessary open ports
- Directory listing enabled
- Debug mode enabled in production
- Misconfigured cloud storage (S3 buckets public)

Consequences

- Full server compromise
- Data leakage
- Unauthorized access

Prevention

- Harden servers
- Disable unused services
- Regular configuration audits
- Apply least privilege

---

7. Insecure Cryptographic Storage

Improper storage of sensitive data using weak or no encryption.

Common Issues

- Storing passwords in plaintext

- Using weak hashing algorithms: MD5, SHA-1
- Poor key management
- Hard-coded encryption keys in code
- No encryption for sensitive fields (credit card numbers, personal data)

Risks

- Data breaches
- Identity theft
- Financial loss

Prevention

- Use strong hashing like bcrypt, PBKDF2, Argon2
- Implement AES-256 for encryption
- Rotate and securely store keys
- Use HSM or secure vaults (HashiCorp Vault, AWS KMS)

Summary / Quick Revision Points

- **Social Engineering** → Attacks human weakness, not systems.
- **Injection** → Untrusted input modifies server-side commands.
- **XSS** → Injecting malicious scripts into webpages.
- **Broken Authentication** → Weak session handling leads to account takeover.
- **CSRF** → Forces user to perform unintended actions.
- **Security Misconfiguration** → Default or weak settings.
- **Insecure Cryptographic Storage** → Weak/no encryption of sensitive data.