Theoretical Knowledge

1. Advanced Vulnerability Exploitation

1.1 Exploit Chaining

Definition

Exploit chaining is the practice of combining **multiple vulnerabilities** in sequence to achieve a more powerful attack outcome.

- A single vulnerability may provide limited access.
- When **chained together**, low- or medium-severity flaws can escalate into **full compromise**.

Why It Matters

- Security teams often underestimate “minor” bugs.
- Attackers combine these small flaws into **multi-stage attack paths**.
- Real-world APTs (Advanced Persistent Threats) rely on chaining for stealth and persistence.

Stages in a Chained Attack

Stage 1 – Initial Foothold

- Gain a **limited entry point** into the system.
- Example: **XSS vulnerability** on a web application → steal session cookies.

Stage 2 – Privilege Escalation

- Use the foothold to gain **higher privileges.**
- Example: With a stolen session → perform **CSRF attack** as the logged-in user.

Stage 3 – System Compromise

- Turn application-level control into **system-level compromise**.
- Example: Upload a **malicious file (webshell)** → achieve **Remote Code Execution (RCE)**.

Stage 4 – Persistence / Lateral Movement

- Establish a **backdoor** or pivot deeper.
- Example: Move from web server → database → internal network.

Example – CSRF + SQL Injection Chain

1. **CSRF Vulnerability:**
    a. Exploited to force an authenticated admin to unknowingly execute a request.
    b. Example: Change admin's profile to inject SQL payload.
2. **SQL Injection Vulnerability:**
    a. Exploited via the crafted request.
    b. Extract **database credentials** or drop a **malicious webshell**.
3. **Chained Outcome:**
    a. From a "simple CSRF" → full **database compromise + RCE**.
    b. Attack escalates far beyond the original bug scope.

Real-World Examples

- **Facebook (2013):**
    - Bug bounty report showed a chain of **XSS + CSRF** → **account takeover**.
- **SolarWinds Attack:**
    - Supply chain compromise → code execution → credential theft → domain-wide persistence.
- **EternalBlue + DoublePulsar:**
    - Buffer overflow exploit (SMBv1) chained with a persistent backdoor.

Key Defenses

- **Defense-in-Depth:** Patch even "low" severity bugs.
- **Threat Modeling:** Assume attackers will chain vulnerabilities.
- **Monitoring:** Look for suspicious sequences, not isolated events.

1.2 Custom Exploit Development

Definition: Adapting or creating exploits (PoCs) to work on specific environments when public ones fail.

Stages:

1. **Recon & Profiling** → Identify OS, version, mitigations (ASLR, DEP).
2. **Analysis** → Reverse engineer code, debug crashes, find exploit primitives.
3. **Adapt/Develop** → Modify PoCs (offsets, payloads) or design new approach.
4. **Bypass Mitigations** → Use concepts like ROP, info leaks, encoding.

5. **Testing & Reporting** → Validate in lab, document proof, provide defenses.

Example (Conceptual): Heap spray in browsers → shaping memory for reliable exploitation.

Defenses: Patch systems, enable ASLR/DEP, input validation, EDR detection, least privilege.

1.3 Bypassing Defenses

Definition: Techniques to evade modern security mechanisms (ASLR, DEP, WAFs) that block exploits.

Stages:

1. **Identify Defenses** → Detect protections (memory randomization, stack canaries, filters).
2. **Develop Strategy** → Choose bypass (ROP, heap spray, shellcode encoding).
3. **Exploit Execution** → Chain gadgets, use polymorphic/obfuscated payloads.
4. **Validation** → Test reliability across reboots/versions.

Example (Conceptual): Using ROP chain to bypass DEP and execute arbitrary code.

Defenses: Patch software, enable modern mitigations (CFG, SMEP), use WAFs with anomaly detection, behavioral EDR

Outcome: Ability to craft, adapt, and chain sophisticated exploits while avoiding detection and maintaining persistence.

2. API Security Testing

2.1 API Vulnerabilities

Definition: Weaknesses in Application Programming Interfaces (APIs) that attackers exploit to gain unauthorized access, manipulate data, or disrupt services.

Core Concept:

- **OWASP API Top 10** → Industry standard list of the most critical API security risks.
    - Example: **A01:2023 – Broken Object Level Authorization (BOLA)** → Attackers manipulate object IDs to access data they shouldn't.

Example:

- Exploiting weak API tokens → attacker modifies a user ID in an API call to fetch another user's private data.

Key Risks:

1. Unauthorized data exposure.
2. Privilege escalation via API endpoints.
3. Abuse of weak authentication/authorization checks.

Outcome:
Attackers can steal sensitive data, disrupt business logic, or compromise entire systems through insecure APIs.

2.2 Testing Techniques

Definition: Methods to identify, validate, and exploit vulnerabilities in applications, APIs, and systems.

Core Concept:

- Use a **combination of manual & automated testing** to uncover flaws.
  - **Manual Testing (Burp Suite):**
    - API endpoint enumeration
    - Parameter tampering
    - Authentication/authorization testing
  - **Automated Testing (Postman, OWASP ZAP, custom scripts):**
    - Fuzzing API parameters
    - Automated request replay
    - Response analysis for anomalies

Example:

- Using Burp Suite to enumerate hidden API endpoints.
- Using Postman to fuzz inputs and detect improper error handling.

Key Risks Found:

1. Broken object-level authorization.
2. Input validation flaws.
3. Data exposure via misconfigured endpoints.

Outcome:
Ensures security gaps are identified early, improving resilience against API and web attacks.

2.3 Rate Limiting and Injection

Definition: Techniques attackers use to bypass request throttling and exploit injection flaws in APIs or applications.

Core Concept:

- **Rate Limiting Bypass:**
  - Manipulating headers (e.g., `X-Forwarded-For`) to appear as different users/IPs.
  - Using multiple API keys, TOR, or botnets to evade restrictions.
- **Injection Attacks:**
  - Inserting malicious payloads into queries (SQLi, NoSQLi, GraphQLi).
  - Exploiting poor input sanitization to extract or modify data.

Example:

- **Bypassing Rate Limits:** Attacker floods login endpoint by rotating IP headers, enabling brute force.
- **GraphQL Injection:** Sending crafted queries (`__schema` introspection) to dump database contents.

Key Risks:

1. Credential stuffing & brute force attacks.
2. Unauthorized data access via GraphQL or NoSQL injection.
3. Service disruption from abuse of unrestricted endpoints.

Outcome:
If unmitigated, attackers can brute-force credentials, extract sensitive data, or pivot deeper into systems.

3. Privilege Escalation and Persistence

3.1 Privilege Escalation

Definition: The process of gaining higher-level permissions (e.g., root, SYSTEM, admin) than originally assigned.

Core Concept:

- **Vertical Escalation** → From low-privileged user to high-privileged (e.g., root).
- **Horizontal Escalation** → Accessing another user's resources without increasing privileges.

Techniques:

1. **Kernel Exploits** → Exploit vulnerabilities in the OS kernel to gain root access.

2. **Misconfigured Services** → Abuse weak service configs (e.g., running as root/admin).
3. **Weak File/Folder Permissions** → Modify scripts, binaries, or configs executed by privileged users.
4. **SUID/SGID Binaries** → Exploit special permission binaries in Linux to escalate privileges.

Example:

- Exploiting a misconfigured **SUID binary** (`/usr/bin/nmap`) to execute commands as root.

Key Risks:

1. Full system compromise.
2. Persistence with high-level backdoors.
3. Ability to disable security tools and cover tracks.

Outcome:
Privilege escalation often turns a limited compromise into **complete system takeover**.

3.2 Persistence Mechanisms:

Definition: Techniques attackers use to maintain long-term access to a compromised system, even after reboots or user logouts.

Core Concept:

- Persistence ensures attackers don't lose access once the initial exploit is closed or detected.
- Can be **OS-level, application-level, or service-level.**

Techniques:

1. **Linux:**
    a. **Cron Jobs** → Schedule malicious scripts to run automatically.
    b. **Backdoored Binaries** → Replace common binaries (e.g., ssh, sudo) with trojanized versions.
    c. **SSH Keys** → Plant attacker's public key in `~/.ssh/authorized_keys`.
2. **Windows:**
    a. **Registry Keys** → Add entries in `HKCU\Software\Microsoft\Windows\CurrentVersion\Run` for persistence.
    b. **Scheduled Tasks** → Automatically run malware at startup.
    c. **Malicious Services/Drivers** → Install services that restart on boot.

Example:

- Attacker adds a **registry run key** in Windows to ensure malware executes every reboot.

Key Risks:

1. Long-term hidden access.
2. Difficult detection and removal.
3. Used in APTs (Advanced Persistent Threats) to survive cleanup efforts.

Outcome:
Persistence turns a one-time compromise into **continuous attacker presence**, enabling data theft or sabotage over time.

3.3 Living-off-the-Land (LotL)

Definition: A technique where attackers abuse trusted, built-in system tools instead of introducing new malware, making their activities stealthier.

Core Concept:

- Avoids detection by **blending malicious actions with normal system operations.**
- Relies on binaries, scripts, and frameworks already present in the OS (a.k.a. **LOLBins/LOLScripts**).

Techniques:

1. **Windows:**
   a. **PowerShell** → Run encoded payloads or download malware.
   b. **WMI (Windows Management Instrumentation)** → Remote execution, persistence, and reconnaissance.
   c. **MSHTA / Certutil** → Download & execute payloads under trusted processes.
2. **Linux:**
   a. **Bash / Cron / Systemd** → Abuse job scheduling or service configs.
   b. **Netcat / Curl / Wget** → Create reverse shells or exfiltrate data.
   c. **Sudo Misuse** → Execute commands via trusted binaries.

Example:

- Attacker uses **PowerShell** to download and execute a reverse shell in memory, avoiding disk artifacts.

Key Risks:

1. Bypasses traditional AV/EDR since tools are legitimate.
2. Harder forensic detection due to lack of new binaries.
3. Enables escalation, persistence, and lateral movement stealthily.
4. **Stealth & Evasion** → Avoid detection while maintaining control (hide processes, use memory-only payloads).

Outcome:
LotL techniques allow attackers to stay **low-profile**, evade defenses, and operate for long periods without raising alerts.

Outcome:
Ability to escalate privileges, survive reboots, and operate undetected—turning a foothold into a **deep, long-term system compromise**

4. Network Protocol Attacks

4.1 Protocol Exploitation

Definition: The abuse of weaknesses in network protocols to gain unauthorized access, harvest credentials, or disrupt services.

Core Concept:

- Many protocols (SMB, DNS, SNMP, FTP, Telnet) were designed before modern security standards.
- Attackers exploit weak authentication, misconfigurations, or trust assumptions in these protocols.

Techniques:

1. **SMB Exploitation**
    a. **SMB Relay Attacks** → Intercept and relay authentication requests to gain access.
    b. **EternalBlue (CVE-2017-0144)** → Exploit SMBv1 buffer overflow for remote code execution.
2. **DNS Exploitation**
    a. **DNS Spoofing/Poisoning** → Redirect traffic to malicious servers.
    b. **DNS Tunneling** → Hide C2 communication inside DNS queries.
3. **SNMP Exploitation**
    a. Exploit default or weak community strings (`public`, `private`).
    b. Use SNMP to dump system configs, user accounts, and network topology.

Example:

- **SMB Relay Attack** → Capture NTLM hashes during authentication and reuse them to authenticate as the victim, enabling credential harvesting and lateral movement.

Key Risks:

1. Credential theft & lateral movement.
2. Redirection of legitimate traffic to malicious endpoints.
3. Full remote code execution (e.g., EternalBlue).

Outcome:
Protocol exploitation can escalate a single foothold into **widespread network compromise** due to trust relationships and legacy services.

4.2 Man-in-the-Middle (MitM)

Definition: An attack where the adversary secretly intercepts, relays, or alters communication between two parties without their knowledge.

Core Concept:

- Exploits weaknesses in **network protocols, trust models, or encryption.**
- Enables attackers to steal credentials, inject malicious payloads, or manipulate traffic.

Techniques:

1. **ARP Spoofing**
   a. Trick local network devices into sending traffic through the attacker's machine.
   b. Common in LAN attacks.
2. **DNS Poisoning**
   a. Redirect traffic by altering DNS responses.
   b. Victim believes they're visiting a legitimate domain but lands on a malicious one.
3. **SSL Stripping**
   a. Downgrade HTTPS connections to HTTP, allowing traffic to be read in plaintext.
   b. Exploits sites misconfigured for strict HTTPS.

Example:

- Attacker performs **ARP spoofing** to intercept credentials sent over HTTP within a corporate LAN.

Key Risks:

1. Credential theft (usernames, passwords, tokens).
2. Session hijacking & account takeover.
3. Data manipulation or injection of malicious content.

Outcome:
MitM attacks allow attackers to **intercept and alter sensitive communications**, breaking confidentiality and integrity of data in transit.

4.3 Protocol Misconfigurations

Definition: Security weaknesses caused by improper or outdated configurations in network protocols, leaving them vulnerable to exploitation.

Core Concept:

- Many default protocol settings prioritize **functionality over security**.
- Attackers exploit unencrypted traffic, weak authentication, or legacy versions.

Techniques / Examples:

1. **Telnet (Unencrypted Login):** Credentials sent in plaintext, easily captured via sniffing.
2. **SMBv1 (Outdated Protocol):** Vulnerable to attacks like **EternalBlue**, enabling RCE.
3. **FTP (Plaintext File Transfer):** No encryption; allows credential harvesting and data theft.
4. **SNMP (Default Community Strings):** “public/private” often unchanged, enabling device takeover.
5. **HTTP (No TLS):** Susceptible to credential theft via sniffing or MitM attacks.

Example:

- A company leaves **Telnet enabled** for remote admin; attacker sniffs credentials and gains access.

Key Risks:

1. Credential theft via plaintext traffic.
2. Remote code execution from legacy protocols.
3. Exposure of sensitive system and network data.

Outcome:
Misconfigured protocols create **low-hanging fruit** for attackers, often leading to full system compromise with minimal effort.

1. **Defense Awareness** → Understand how secure versions, encryption, and monitoring can mitigate risks.

Outcome:
Ability to exploit insecure protocols for **credential harvesting, lateral movement, or data theft**, while recognizing mitigations to defend networks.

5. Mobile Application Penetration Testing

5.1 Mobile Vulnerabilities

Definition: Security flaws in mobile applications or platforms (Android, iOS) that expose sensitive data or allow unauthorized access.

Core Concept:

- Guided by **OWASP Mobile Top 10**, which outlines the most critical risks in mobile security.

Techniques / Examples:

1. **M1: Improper Platform Usage** → Misuse of platform features (e.g., Android intents, iOS TouchID).
    a. Example: Insecure use of Android intent allows data leakage between apps.
2. **M2: Insecure Data Storage** → Sensitive data stored in plaintext on device or SD card.
    a. Example: Extracting API keys or tokens from an unencrypted SQLite database in an Android APK.
3. **M3: Insecure Communication** → Lack of TLS or weak certificate validation.
    a. Example: Attacker intercepts mobile app traffic with a MitM proxy.
4. **M4: Insecure Authentication** → Weak session tokens or missing multi-factor checks.

Example:

- Reverse-engineering an Android APK to recover hardcoded credentials or encryption keys.

Key Risks:

1. Credential theft and session hijacking.
2. Sensitive data exposure (tokens, payment info, PII).
3. Remote compromise of mobile devices.

Outcome:
Mobile vulnerabilities can lead to **data breaches, fraud, or device compromise**, often at scale due to widespread app usage.

5.2 Testing Techniques

Definition: Methods to detect mobile application vulnerabilities through code analysis and runtime testing.

Core Concept:

- Use a mix of **Static Analysis** (code-level review) and **Dynamic Testing** (runtime behavior analysis).

Techniques / Tools:

1. **Static Analysis (SAST):**
    a. Tool: **MobSF (Mobile Security Framework)**
    b. Checks for insecure code patterns, hardcoded secrets, weak crypto, misconfigurations.

   c. Example: Detecting hardcoded API keys in Android source code.
2. **Dynamic Testing (DAST):**
   a. Tool: **Frida** (runtime instrumentation framework).
   b. Used for **runtime manipulation** → bypassing root detection, tampering with API calls.
   c. Example: Hooking functions in a banking app to bypass 2FA checks.
3. **Hybrid Approach:**
   a. Combine MobSF + Frida with network traffic analysis (Burp Suite, mitmproxy).
   b. Example: Intercepting unencrypted API calls from mobile app during execution.

Key Risks Identified:

- Hardcoded secrets
- Insecure data storage
- Weak authentication checks
- API communication leaks

Outcome:
Testing techniques reveal both **design flaws (static)** and **runtime weaknesses (dynamic)**, ensuring complete mobile app security coverage.

5.3 Secure Mobile Design

Definition: Techniques and best practices to protect mobile applications from common vulnerabilities and attacks.

Core Concept:

- Focus on **data protection, code integrity, and runtime security** to reduce attack surface.

Techniques / Mitigations:

1. **Secure Storage:**
   a. Use encrypted storage for sensitive data (e.g., Keychain on iOS, EncryptedSharedPreferences on Android).
2. **Code Obfuscation:**
   a. Obfuscate class names, methods, and strings to prevent reverse engineering (e.g., ProGuard, DexGuard).
3. **Runtime Checks:**
   a. Detect rooted/jailbroken devices, tampering, or debugging attempts.
   b. Prevent sensitive actions when integrity checks fail.
4. **Secure Communication:**
   a. Enforce TLS, certificate pinning, and input validation.

Example:

- A banking app encrypts all sensitive data, obfuscates code, and refuses to run on rooted devices.

6. Comprehensive Reporting and Remediation

PTES Reporting Template Structure

The module focuses on creating **professional, actionable reports** for both executive leadership and technical teams, ensuring clarity, traceability, and prioritization.

1. Executive Summary (For Leadership and Oversight)

Purpose:
Provide a **high-level view of business risk** and overall security posture without delving into technical minutiae.

Subsections:

1. **Background**
    a. Why the assessment was conducted, systems in-scope, type of data processed, key risks considered.
    b. **Example:**
    "<Client> engaged <Tester> to assess external-facing systems that store confidential customer data and intellectual property."
2. **Overall Posture**
    a. Narrative on how well defenses worked.
    b. Highlight systemic issues (e.g., weak patch management) vs. symptomatic issues (e.g., one unpatched server).
3. **Risk Ranking/Profile**
    a. Overall risk score using CVSS, DREAD, FAIR, or custom scoring.
    b. **Example:** "Overall Risk Rating = 7 (Elevated)."
4. **General Findings**
    a. Summarize vulnerabilities and attack outcomes using visuals (charts, graphs, tables).
    b. Count of critical/high/medium findings.
    c. Common root causes: misconfigurations, poor credential hygiene, weak patching.
5. **Recommendation Summary**
    a. High-level guidance on remediation.
    b. Prioritize fixes based on risk and effort required.
6. **Strategic Roadmap**
    a. Prioritized, time-based remediation plan (short-, mid-, long-term).
    b. Aligned with business objectives and organizational threat model.

2. Technical Report (For IT / Security Teams)

Purpose:
Provide **in-depth technical details** about scope, methodology, findings, exploit paths, and remediation. Designed for engineers, sysadmins, and IT teams.

Subsections:

1. **Introduction**
   a. Participants: Client + Tester teams
   b. Scope: Systems, apps, networks, cloud, endpoints
   c. Objectives: Confidentiality, integrity, availability
   d. Methodology & Threat Model: PTES, MITRE ATT&CK, STRIDE
2. **Information Gathering**
   a. **Passive Intelligence:** DNS records, Google dorks, public data
   b. **Active Intelligence:** Nmap scans, service discovery, network mapping
   c. **Corporate Intelligence:** Org structure, partnerships, market position
   d. **Personnel Intelligence:** Employee info, leaked credentials
3. **Vulnerability Assessment**
   a. Techniques: Automated scanners + manual testing
   b. Classification:
      i. Technical vulnerabilities (SQLi, XSS, RCE, mapped to OSI layers)
      ii. Logical vulnerabilities (business logic flaws, insecure workflows)
4. **Exploitation & Vulnerability Confirmation**
   a. Steps to confirm vulnerabilities
   b. Exploitation timeline: hosts, times, methods
   c. Attack methods: Direct exploitation, phishing, client/browser-side
   d. Evidence: Screenshots, logs, proof of shell/system access
   e. Access levels achieved: User, admin, root
   f. Remediation & mitigating controls (temporary + long-term)
5. **Post-Exploitation**
   a. How vulnerabilities were leveraged for real-world risk:
      i. Privilege escalation
      ii. Access to sensitive/critical data
      iii. Domain or business system compromise
      iv. Exfiltration, persistence, and pivoting
6. **Countermeasure Effectiveness**
   a. Which defenses worked as expected (firewalls, WAF, IDS/IPS)
   b. Which were bypassed
   c. Incident response effectiveness
7. **Risk & Exposure Analysis**
   a. Likelihood of exploitation
   b. Attacker skill required
   c. Control strength (weak/medium/strong)
   d. Potential losses: financial, reputational, regulatory
   e. Root cause analysis: Process gaps, misconfigurations, missing patches
8. **Conclusion**

a. Restate key outcomes
b. Forward-looking security guidance
c. Highlight positives (what’s working well)
d. Encourage ongoing security improvements aligned with roadmap