	P.E.S. COLLEGE OF ENGINEERING **(AN AUTONOMOUS INSTITUTE)** **CHH. SAMBHAJINAGAR-431002** **Regular Winter Examination – 2025** **Course: F.Y.M. Tech. Branch : CSE (Cyber security) Semester : I** **Subject Code & Name: MTPESCS105TB Mobile Application Security** **Max Marks: 60 Date: Duration: 3 Hr.**		
	Instructions to the Students: *1. All the questions are compulsory.* *2. The level of question/expected answer as per OBE or the Course Outcome (CO) on which the question is based is mentioned in ( ) in front of the question.* *3. Use of non-programmable scientific calculators is allowed.* *4. Assume suitable data wherever necessary and mention it clearly.*		
		(Level/CO)	Marks
Q. 1	**Solve Any one of the following.**		
A)	**1.Explain the evolution of mobile communication platforms from GSM → 5G with respect to security enhancements.** Mobile communication platforms have evolved significantly from GSM (2G) to 5G, not only to improve data rates and latency but also to address growing security threats. Early mobile systems focused mainly on voice communication with basic security, whereas modern networks support high-speed data, IoT, cloud integration, and critical services, demanding strong authentication, encryption, privacy, and integrity mechanisms. Each generation has introduced enhanced security features to overcome the limitations of previous technologies. **1. GSM (2G) – Second Generation** **Security Features** GSM was the first digital cellular system and introduced basic security mechanisms mainly to protect voice communication. It uses SIM-based authentication, where a secret key (Ki) stored in the SIM is used for authentication. Encryption algorithms such as A5/1 and A5/2 are used to protect data over the air interface. **Limitations** GSM provides one-way authentication, where the network authenticates the user but the user cannot authenticate the network. This makes GSM vulnerable to fake base station attacks (IMSI catchers). Encryption is applied only between the mobile device and the base station, not end-to-end. Weak encryption algorithms and lack of integrity protection make GSM insecure against modern attacks. **2. UMTS (3G) – Third Generation** Security Enhancements over GSM UMTS significantly improved security by introducing mutual authentication, where both the user and the network authenticate each other. Stronger encryption algorithms such as KASUMI are used. UMTS also introduced integrity protection to prevent modification of signaling messages. **Additional Improvements** Temporary identities are used to protect user identity and prevent tracking. Key management and authentication procedures are more robust than GSM. These enhancements reduce the risk of impersonation and replay attacks. **Remaining Issues** Although more secure than GSM, 3G still relies on centralized network ar-	**CO1**	**6**

chitecture and is vulnerable to signaling attacks and legacy fallback attacks.

3. LTE (4G) – Fourth Generation

Security Advancements

LTE introduced a packet-switched, IP-based architecture, requiring stronger security. It uses the EPS-AKA (Evolved Packet System Authentication and Key Agreement) protocol for authentication. Strong encryption algorithms such as AES and SNOW 3G are used.

Key Security Features

LTE provides better protection for user identity using temporary identifiers (GUTI). Integrity protection is applied to both control-plane and user-plane data. Secure key hierarchy and separation of access and core network keys improve overall security.

Challenges

Despite improvements, LTE is vulnerable to Denial of Service (DoS) attacks, rogue base stations, and protocol-level attacks due to backward compatibility with older networks.

4. 5G – Fifth Generation

Advanced Security Architecture

5G introduces a service-based architecture and supports applications such as IoT, autonomous vehicles, and smart cities, requiring very high security. It uses 5G-AKA and EAP-AKA' protocols for authentication. Unlike earlier generations, 5G provides privacy protection for permanent subscriber identity (SUPI) using encryption, preventing IMSI catching.

Enhanced Security Features

5G supports end-to-end security, stronger encryption algorithms, and improved integrity protection. It introduces network slicing security, ensuring isolation between different virtual networks. Secure communication between network functions is ensured using TLS and service authentication.

New Capabilities

5G also supports edge computing security, improved key management, and better protection against replay, impersonation, and man-in-the-middle attacks.

2.Discuss common Bluetooth security vulnerabilities such as BlueBorne and BlueSnarfing.

6

Bluetooth is a widely used short-range wireless communication technology in mobile devices, IoT gadgets, wearables, and automotive systems. Although Bluetooth provides convenience and low power consumption, it also introduces several security vulnerabilities due to improper authentication, weak pairing mechanisms, and implementation flaws. Two well-known Bluetooth attacks are **BlueBorne** and **BlueSnarfing**, which exploit weaknesses in Bluetooth protocols and device configurations.

1. BlueBorne Attack

BlueBorne is a critical Bluetooth vulnerability that allows attackers to gain unauthorized access to devices without pairing or user interaction. This attack exploits flaws in the Bluetooth implementation of operating systems such as Android, Windows, Linux, and iOS. Since Bluetooth is usually enabled by default, devices become vulnerable even when they are not actively connected to another device.

How the Attack Works

An attacker scans for Bluetooth-enabled devices within range. Once a vulnerable device is found, the attacker exploits protocol-level flaws to execute malicious code or access system resources. BlueBorne attacks do not require pairing, making them extremely dangerous.

Impact

	BlueBorne can lead to remote code execution, data theft, man-in-the-middle attacks, and device takeover. Since the attack does not leave obvious traces, it is difficult to detect. It can spread rapidly in crowded areas such as offices, airports, or conferences. **Security Weaknesses Exploited** • Lack of proper validation in Bluetooth protocol implementation • Always-on Bluetooth interfaces • Absence of user authentication **2. BlueSnarfing Attack** BlueSnarfing is a Bluetooth attack that allows an attacker to steal sensitive information from a target device without the owner's knowledge. This includes contacts, messages, emails, and calendar data. The attack typically exploits weak pairing mechanisms or misconfigured Bluetooth services. **How the Attack Works** The attacker connects to a Bluetooth device that is set to discoverable or improperly secured. By exploiting Object Exchange (OBEX) protocol vulnerabilities, the attacker accesses stored files and personal data without authorization. **Impact** BlueSnarfing leads to privacy violations and identity theft. Attackers can extract confidential data and use it for social engineering or further cyber attacks. **Security Weaknesses Exploited** • Devices left in discoverable mode • Weak or no authentication • Improper implementation of OBEX services		
B)	**Describe common cellular network security issues in 3G, 4G, and 5G architectures.** Cellular network architectures have evolved from 3G to 5G to support high-speed data, low latency, and massive connectivity. Along with these advancements, new security challenges have emerged due to increased complexity, IP-based communication, virtualization, and backward compatibility. Each generation has its own set of security issues related to authentication, privacy, signaling, and network infrastructure. **1. Security Issues in 3G (UMTS) Architecture** **Authentication and Identity Issues** Although 3G introduced **mutual authentication**, it still exposes user identity in some scenarios, especially during inter-network roaming. Temporary identifiers can be compromised, enabling user tracking. **Signaling Attacks** 3G networks rely heavily on signaling protocols such as **SS7**, which lack strong authentication. Attackers can exploit SS7 vulnerabilities to intercept calls, track user location, or perform call forwarding attacks. **Denial of Service (DoS) Attacks** Attackers can overload network elements like Node B or Radio Network Controller (RNC) by sending excessive signaling requests, causing service disruption. **Backward Compatibility Threats** 3G devices often fall back to 2G networks in low-coverage areas, exposing them to GSM-level attacks such as fake base stations. **2. Security Issues in 4G (LTE) Architecture** **Rogue Base Station Attacks**	**CO1**	**12**

	LTE is vulnerable to fake eNodeB attacks, where attackers deploy rogue base stations to intercept traffic, downgrade encryption, or perform man-in-the-middle attacks. **IMSI Catching and Privacy Leakage** Despite improvements, subscriber identity can still be exposed during certain procedures. IMSI catchers can be used to track users and monitor activity. **Signaling Storms and DoS Attacks** LTE's IP-based architecture is vulnerable to signaling storms caused by malicious apps or attackers, leading to network congestion and denial of service. **Protocol Vulnerabilities** Weaknesses in LTE protocols such as **NAS (Non-Access Stratum)** can be exploited to cause authentication failures, forced detachments, or service denial. **3. Security Issues in 5G Architecture** **Increased Attack Surface** 5G uses a **service-based architecture (SBA)** with virtualization and cloud-native components. This increases the attack surface and exposes APIs to cyber threats. **Network Slicing Risks** Network slicing allows multiple virtual networks to run on shared infrastructure. Improper isolation between slices can lead to data leakage or cross-slice attacks. **Edge Computing Vulnerabilities** 5G heavily relies on **Mobile Edge Computing (MEC)**. If edge nodes are compromised, attackers can access sensitive data or disrupt latency-critical services. **Legacy Network Interworking** 5G still supports interoperability with 4G and 3G networks. Attackers can exploit this backward compatibility to force devices to downgrade to less secure networks. **IoT and Massive Device Security** 5G supports massive IoT deployments. Poorly secured IoT devices can be used as entry points for attacks or as part of botnets.		
Q.2	**Solve Any one of the following.**		
A)	**Discuss the OWASP Mobile Top 10 security risks with examples.** The **OWASP Mobile Top 10** is a globally recognized list that identifies the most critical security risks affecting mobile applications. These risks arise due to insecure coding practices, weak authentication, improper data storage, and poor communication security. Understanding these risks helps developers design secure mobile applications and protect sensitive user data from cyber attacks. **1. M1 – Improper Platform Usage** **Description** This risk occurs when mobile platform features such as permissions, intents, Touch ID, or keychain are used incorrectly or insecurely. **Example** An Android app exposes sensitive data using exported activities without	**CO2**	**12**

proper permission checks.

Solution

- Follow platform security guidelines
- Restrict permissions and exported components
- Use secure APIs correctly

2. M2 – Insecure Data Storage

Description

Sensitive data is stored locally on the device in an insecure manner without encryption.

Example

Storing passwords or tokens in plaintext in SharedPreferences or SQLite database.

Solution

- Encrypt sensitive data
- Use secure storage mechanisms (Android Keystore, iOS Keychain)
- Avoid storing unnecessary sensitive information

3. M3 – Insecure Communication

Description

Data is transmitted over insecure channels without proper encryption.

Example

Mobile app sends login credentials over HTTP instead of HTTPS.

Solution

- Enforce HTTPS with TLS
- Implement certificate pinning
- Avoid insecure Wi-Fi communication

4. M4 – Insecure Authentication

Description

Weak authentication mechanisms allow attackers to bypass login controls.

Example

Using simple PINs or lacking session timeout in a banking app.

Solution

- Implement strong authentication (multi-factor authentication)
- Enforce session expiration
- Protect authentication APIs

5. M5 – Insufficient Cryptography

Description

Weak or outdated cryptographic algorithms are used to protect sensitive data.

Example

Using MD5 or SHA-1 for password hashing.

Solution

- Use strong cryptographic algorithms (AES, SHA-256)
- Use platform-provided cryptographic libraries
- Properly manage encryption keys

6. M6 – Insecure Authorization

Description

Authorization checks are missing or improperly implemented.

Example

A user can access admin features by modifying API request parameters.

Solution

- Enforce role-based access control
- Validate user permissions on the server side
- Do not rely on client-side authorization

7. M7 – Client Code Quality Issues

	Description Poor coding practices introduce vulnerabilities such as buffer overflows or injection flaws. **Example** App crashes due to improper input validation, allowing code injection. **Solution** • Validate all user inputs • Perform secure code reviews • Use static and dynamic analysis tools **8. M8 – Code Tampering** **Description** Attackers reverse-engineer or modify the app code to bypass security controls. **Example** Removing license checks from a paid application using APK modification. **Solution** • Use code obfuscation • Implement runtime integrity checks • Detect rooted or jailbroken devices **9. M9 – Reverse Engineering** **Description** Attackers analyze the app binary to extract sensitive information or logic. **Example** Hardcoded API keys found using reverse engineering tools. **Solution** • Avoid hardcoding secrets • Obfuscate code and strings • Use server-side validation **10. M10 – Extraneous Functionality** **Description** Debug code, hidden backdoors, or test APIs are left in the production app. **Example** Hidden admin APIs accessible without authentication. **Solution** • Remove debug features before release • Conduct penetration testing • Follow secure SDLC practices		
B)	**What is API Security for mobile applications? Discuss problems like Broken Object Level Authorization (BOLA) with examples.** Modern mobile applications rely heavily on **Application Programming Interfaces (APIs)** to communicate with backend servers for authentication, data storage, and business logic. API security refers to the set of practices and mechanisms used to protect these APIs from unauthorized access, misuse, and attacks. Weak API security can lead to serious vulnerabilities such as data leakage, account takeover, and unauthorized operations. One of the most critical API security issues is **Broken Object Level Authorization (BOLA)**. **API Security in Mobile Applications** API security ensures that only authorized users and devices can access backend resources through APIs. Mobile APIs often expose sensitive operations such as fetching user data, processing payments, or updating profiles. Since mobile apps communicate over public networks, APIs must enforce	**CO2**	**12**

strong **authentication, authorization, encryption, and input validation**. Unlike web applications, mobile APIs are frequently reverse-engineered, making them attractive targets for attackers.

Common API Security Challenges

- Inadequate authentication and authorization
- Excessive data exposure in API responses
- Lack of rate limiting
- Weak input validation
- Broken access control mechanisms

Among these, **Broken Object Level Authorization (BOLA)** is one of the most exploited vulnerabilities.

Broken Object Level Authorization (BOLA)

Explanation

BOLA occurs when an API does not properly verify whether the authenticated user has permission to access a specific object or resource. APIs often identify objects using parameters such as user IDs, account numbers, or order IDs. If authorization checks are missing or weak, attackers can manipulate these identifiers to access data belonging to other users.

How BOLA Attack Works

1. A user logs into a mobile application and captures an API request.
2. The API request contains an object identifier (e.g., user_id=101).
3. The attacker modifies the identifier (e.g., user_id=102).
4. If the API does not verify ownership, it returns another user's data.

Example of BOLA Attack

Scenario

A mobile banking app provides an API endpoint:
GET /api/accounts/{account_id}
Authenticated user A has account_id = 5001.
By changing the request to account_id = 5002, user A can access user B's account details if the API does not enforce object-level authorization.

Impact

- Unauthorized access to sensitive data
- Privacy violation
- Financial fraud
- Regulatory compliance issues

Why BOLA Is Common in Mobile APIs

- APIs trust client-supplied object IDs
- Developers rely only on authentication, not authorization
- Lack of server-side ownership validation
- Extensive use of REST APIs with predictable IDs

Mitigation and Solutions for BOLA

1. Enforce Object-Level Authorization

Always verify that the authenticated user owns or is authorized to access the requested object.

2. Use Indirect Object References

Avoid exposing direct database IDs. Use random or hashed identifiers.

3. Server-Side Access Control

Never rely on client-side checks. Perform all authorization on the server.

4. Implement Role-Based Access Control (RBAC)

	Ensure users can only access resources permitted by their role. **5. API Security Testing** Perform regular penetration testing and automated security testing to identify BOLA vulnerabilities. **6. Use API Gateways** API gateways can enforce authorization rules, rate limiting, and monitoring.		
Q. 3	**Solve Any one of the following.**		
A)	**1.Differentiate between Static Analysis and Dynamic Analysis in the context of mobile application security.**	**CO3**	**6**

Feature	Static Analysis	Dynamic Analysis
Definition	Code inspection without execution	Testing app behavior at runtime
Timing	Before app execution	During app execution
Scope	Source code, binaries, configs	App behavior, network traffic, runtime data
Vulnerabilities Detected	Hardcoded credentials, insecure API use, poor encryption	Runtime logic flaws, authentication bypass, data leakage
Advantages	Early detection, full code coverage	Detects runtime vulnerabilities, realistic testing
Disadvantages	False positives, cannot detect runtime issues	Cannot cover all paths, requires test environment

2.Explain the process of decompiling Android APKs and the role of tools like jadx and apktool. **6**

An **Android APK (Android Package Kit)** is a compressed file containing all resources, code, and configuration required for an Android application. **Decompiling** is the process of converting the APK's compiled code and resources back into a human-readable or editable form. This process is widely used in **mobile security testing, malware analysis, and reverse engineering** to identify vulnerabilities, inspect code behavior, or recover lost source code.

Decompile Android APKs

- Analyze security vulnerabilities such as **hardcoded credentials or insecure API keys**
- Inspect **app logic and behavior**
- Perform **malware analysis**
- Recover **lost source code for maintenance**

Structure of an APK

An APK file contains:

1. **AndroidManifest.xml** – App metadata, permissions, activities
2. **classes.dex** – Compiled Dalvik bytecode
3. **res/** – App resources like images, layouts, and strings
4. **lib/** – Native libraries (C/C++)
5. **META-INF/** – Signature and certificates

Process of Decompiling Android APKs

Step 1: Extract APK Contents

	• APK is a **ZIP archive**, so it can be unzipped using any archive tool or apktool. • Extracting gives access to resources, manifest, and compiled code. **Step 2: Decompile Bytecode to Source** • **Android apps use .dex files** (Dalvik Executable). • Tools like **JADX** convert .dex bytecode into **readable Java source code**. • Example: • jadx-gui app.apk This opens a GUI showing reconstructed Java classes for analysis. **Step 3: Decompile Resources** • APK resources (XML files, layouts) are compiled in binary format. • **Apktool** can decode these resources into **human-readable XML**. • Example: • apktool d app.apk Output includes: o AndroidManifest.xml o res/ directory with layouts, strings, and images **Step 4: Analyze Decompiled Code** • Inspect Java code for vulnerabilities such as: o Hardcoded credentials o Insecure API usage o Improper cryptography • Analyze AndroidManifest.xml for **permissions and exported components** **Role of Tools** **1. JADX** • Converts .dex bytecode to **Java source code** • Supports GUI and CLI modes • Useful for **security auditing and static analysis** • Does **not handle resources** (requires Apktool for that) **2. Apktool** • Decodes **resources and manifest files** • Rebuilds APK after modifications (recompilation) • Used for **reverse engineering, modifying apps, and testing patches** **Example Workflow** 1. Decompile APK with Apktool to extract XML and resources: 2. apktool d app.apk -o app_decompiled 3. Use JADX to convert classes.dex to Java source code: 4. jadx-gui app.apk 5. Inspect permissions in **AndroidManifest.xml** 6. Analyze source code for vulnerabilities such as: o Hardcoded passwords o Insecure HTTP requests o Weak cryptography **Applications in Mobile Security** • **Penetration Testing**: Identify potential security flaws • **Malware Analysis**: Understand malicious behavior • **Compliance Audits**: Ensure app follows secure coding standards • **Education**: Learn Android app internals		
B)	**Explain dynamic instrumentation techniques using tools like Frida, Xposed, or Objection.**	**CO3**	**12**

Dynamic Instrumentation?

Dynamic instrumentation refers to injecting custom code or hooks into a running application to monitor, modify, or control its behavior. It enables security researchers to:

- Intercept API calls
- Modify function return values
- Bypass authentication and root detection
- Analyze encryption, API calls, and runtime logic

This technique is widely used in **penetration testing, malware analysis, and reverse engineering** of mobile applications.

1. Frida

Overview

Frida is a **dynamic instrumentation toolkit** that allows developers and security analysts to inject JavaScript-based hooks into native and Java methods of Android and iOS applications at runtime.

Working Mechanism

Frida attaches to a running process and intercepts function calls using injected scripts. These scripts can monitor arguments, modify return values, or bypass security checks.

Key Features

- Supports Android and iOS
- Hooks Java and native (C/C++) functions
- Real-time manipulation of app behavior
- No need to modify APK permanently

Example Use Case

- Bypassing SSL pinning by intercepting certificate validation methods
- Monitoring sensitive API calls such as login or encryption functions

Advantages

- Very flexible and powerful
- Works without recompiling the app
- Suitable for live analysis

Limitations

- Requires debugging permissions or rooted/jailbroken devices
- Detection-resistant apps may block Frida

2. Xposed Framework

Overview

Xposed is a framework for Android that allows developers to modify system and application behavior by hooking methods at runtime. It works by replacing parts of the Android runtime (ART).

Working Mechanism

Xposed loads modules during system startup. These modules hook into methods of target apps or the Android framework, allowing modification of behavior globally or per app.

Key Features

- Deep integration with Android system
- Persistent hooks across app restarts
- Supports system-level modifications

Example Use Case

- Bypassing root detection logic
- Modifying app permissions and behavior

Advantages

- Powerful system-wide control
- Persistent hooks

Limitations

	• Requires rooted device
	• Less stealthy than Frida
	• Limited to Android only
	3. Objection
	Overview
	Objection is a **runtime mobile exploration framework** built on top of Frida. It simplifies dynamic instrumentation by providing ready-made commands for common mobile security testing tasks.
	Working Mechanism
	Objection uses Frida under the hood to inject hooks but provides an interactive command-line interface for easier testing.
	Key Features
	• No coding required for basic attacks
	• Built-in commands for:
	  ○ SSL pinning bypass
	  ○ Root/jailbreak detection bypass
	  ○ Runtime inspection
	Example Use Case
	• Quickly bypassing SSL pinning in a mobile banking app
	• Exploring app memory and filesystem
	Advantages
	• Easy to use for beginners
	• Automates common Frida tasks
	Limitations
	• Less flexible than raw Frida scripts
	• Depends on Frida		

Q.4	**Solve Any one of the following.**		
A)	**Describe the malware analysis lifecycle for mobile devices.** **Malware Analysis Lifecycle Stages** **1. Collection / Acquisition** **Objective:** Gather malware samples from suspected devices or networks. **Process:** • Collect APK files from devices, app stores, or third-party sources • Capture network traffic suspected of carrying malware payloads • Isolate the malware in a **secure environment** to prevent accidental spread **Tools:** VirusTotal, AnyRun, Threat intelligence feeds **2. Identification / Triage** **Objective:** Quickly classify and prioritize malware samples. **Process:** • Determine type of malware (trojan, ransomware, spyware) • Identify target platform (Android, iOS) • Compare hashes with known malware databases to check for known variants **Tools:** YARA rules, ClamAV, Mobile malware databases **3. Static Analysis** **Objective:** Examine the malware **without executing it**. **Process:**	**CO4**	**12**

- Decompile APK using tools like **JADX** or **Apktool**
- Inspect **AndroidManifest.xml** for permissions and exported components
- Analyze code for suspicious functions, API calls, hardcoded credentials

Purpose: Understand **potential capabilities** of malware (data exfiltration, SMS fraud, etc.)

4. Dynamic Analysis
Objective: Observe malware behavior **during execution** in a controlled environment.
Process:

- Install malware on an **emulator or isolated device**
- Monitor runtime behavior:
  - Network connections
  - File system modifications
  - API calls
  - Cryptographic operations
- Use tools like **Frida, Objection, Wireshark, or Xposed** for runtime instrumentation

Purpose: Detect **actual effects** on the device and network

5. Behavioral Analysis
Objective: Understand how malware interacts with the device and user.
Process:

- Examine interactions with contacts, SMS, call logs, GPS, and sensors
- Identify persistence mechanisms (auto-start, background services)
- Detect attempts to bypass security controls, such as root detection

Tools: Sandbox environments, dynamic instrumentation tools

6. Impact Assessment
Objective: Evaluate the **potential damage and risks** of malware.
Process:

- Determine data compromised (passwords, personal data)
- Assess financial or reputational impact
- Identify affected systems and apps

Outcome: Classify severity and prioritize response

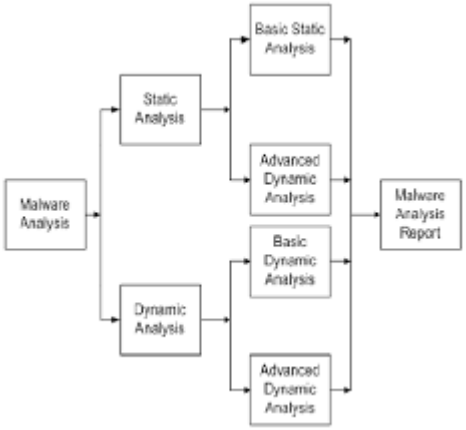
7. Reporting and Mitigation
Objective: Document findings and recommend remediation.
Process:

- Prepare malware analysis reports detailing:
  - Type and behavior of malware
  - Indicators of compromise (IOCs)
  - Recommended mitigation steps
- Share IOCs with threat intelligence communities
- Suggest patches, app removal, or device reset for affected users

Tools: Threat intelligence platforms, SIEM, antivirus updates

8. Prevention and Feedback
Objective: Prevent future infections using analysis insights.
Process:

- Update mobile security policies and antivirus definitions
- Educate users about safe app installation practices

	• Strengthen API and app security based on malware behavior **Diagram: Malware Analysis Lifecycle**  		
B)	**Define mobile malware. Discuss different classes such as spyware, ransomware, Trojans, and adware.** **Definition** **Mobile malware** is software that intentionally performs harmful actions on mobile devices, often without the user's consent, exploiting vulnerabilities in mobile operating systems, applications, or communication protocols. **Classes of Mobile Malware** **1. Spyware** **Definition:** Software designed to secretly monitor and collect data from a mobile device without user consent. **Behavior:** • Tracks SMS, calls, emails, and location • Monitors social media activity • Sends collected data to attackers **Example:** Pegasus spyware can remotely access contacts, messages, and camera/microphone on iOS and Android devices. **Impact:** Privacy violation, identity theft, corporate espionage **Mitigation:** • Install apps from trusted sources only • Use mobile antivirus and anti-spyware apps • Regular OS and app updates **2. Ransomware** **Definition:** Malware that **encrypts user data** or locks the device, demanding a ransom for restoration. **Behavior:** • Locks device screen or encrypts files • Displays a ransom message demanding payment (usually cryptocurrency) • Threatens permanent data loss if ransom is not paid **Example:** Android "Locker" ransomware encrypts internal storage and displays a ransom message. **Impact:** Data loss, financial extortion, service disruption **Mitigation:** • Regular device backups • Avoid clicking suspicious links or installing untrusted apps • Keep security patches updated	**CO4**	**12**

	3. Trojans **Definition:** Malware disguised as a legitimate app to **trick users into installation**, often delivering malicious payloads in the background. **Behavior:** • Exploits user trust to gain access • Can download additional malware • May steal credentials, SMS, or banking information **Example:** "BankBot" Trojan targets Android banking apps, stealing login credentials. **Impact:** Financial fraud, data theft, unauthorized access **Mitigation:** • Install apps only from official app stores (Google Play, App Store) • Use app permission management to restrict sensitive access • Employ mobile threat defense solutions **4. Adware** **Definition:** Software that automatically displays or downloads **advertisements**, often intrusively. While some adware is relatively harmless, aggressive adware can compromise security and performance. **Behavior:** • Displays unwanted ads in apps or browser • May track user behavior and collect analytics without consent • Can slow down device and drain battery **Example:** Some free apps bundle adware that collects user browsing data and shows pop-up ads. **Impact:** Privacy invasion, decreased device performance, potential gateway to other malware **Mitigation:** • Avoid pirated or untrusted apps • Use ad-blockers and privacy-focused apps • Regularly monitor and uninstall suspicious apps		
Q. 5	**Solve Any one of the following.**		
A)	**Explain the mobile forensics process model with each stage in detail.** **Stages of Mobile Forensics Process Model** **1. Identification** **Objective:** Identify potential sources of digital evidence. **Details:** • Determine the types of devices involved (Android, iOS, Windows Mobile) • Identify relevant apps, data storage, communication channels (SMS, call logs, social media) • Assess the **legal scope** for evidence collection • Document device model, serial number, and condition **Example:** Identifying a suspect's smartphone in a cyberfraud case. **2. Preservation** **Objective:** Preserve the original state of the mobile device to prevent evidence tampering. **Details:** • Enable **airplane mode** to prevent remote access	**CO5**	**12**

- Avoid powering off or manipulating the device unnecessarily
- Use **Faraday bags** to block network signals
- Create **forensic images or snapshots** of device memory

Importance: Maintaining **evidentiary integrity** and admissibility in court.

3. Collection / Acquisition

Objective: Extract data from the mobile device in a **forensically sound manner**.

Details:

- Use **physical acquisition** (bit-by-bit copy of flash memory)
- Use **logical acquisition** (export files, messages, contacts)
- Consider **manual extraction** for locked or encrypted devices
- Document extraction process and tools used (Cellebrite UFED, Magnet AXIOM, Oxygen Forensic Suite)

Example: Extracting call logs, SMS, and WhatsApp chat history from an Android phone.

4. Examination / Analysis

Objective: Analyze the collected data to extract **relevant information**.

Details:

- Inspect artifacts like:
  - SMS, call logs, emails
  - GPS/location data
  - Application data (social media, banking apps)
  - Deleted or hidden files
- Recover encrypted or deleted data
- Identify patterns, anomalies, and timestamps relevant to the investigation

Tools: Autopsy, FTK, EnCase, XRY

5. Interpretation

Objective: Understand the context and meaning of the extracted evidence.

Details:

- Correlate data from multiple sources (device, cloud backup, SIM card)
- Establish timelines of user activity
- Identify user intent, access patterns, and possible malicious activity
- Determine which data is admissible in legal proceedings

Example: Linking GPS data to the suspect's location during a crime.

6. Documentation and Reporting

Objective: Prepare a **comprehensive report** for investigators, management, or the court.

Details:

- Record methodology, tools, and procedures used
- Present findings in a **clear, concise, and legally admissible format**
- Include **screenshots, extracted files, and logs**
- Provide recommendations for further investigation or mitigation

Importance: Ensures **chain of custody** and supports legal proceedings.

7. Presentation

Objective: Present evidence in court or to relevant authorities.

Details:

- Explain technical findings in **non-technical terms** for judges and juries

	• Demonstrate the integrity of the evidence • Answer expert queries related to mobile device analysis **Tools:** Forensic reports, dashboards, visualization tools		
B)	**Describe how forensic analysts extract call logs, SMS, location data, and app artifacts from mobile devices.** **General Extraction Approaches** Before extracting specific data, analysts choose an appropriate **data acquisition method** based on device type, OS version, and security controls. **Common Acquisition Methods** • **Logical Extraction** – Extracts user-level data via OS APIs • **File System Extraction** – Accesses full file system structure • **Physical Extraction** – Bit-by-bit copy of device memory • **Manual Extraction** – Viewing and documenting data directly from the device **Tools Used:** Cellebrite UFED, Oxygen Forensic Detective, Magnet AXIOM, XRY **1. Extraction of Call Logs** **Data Stored** Call logs contain: • Incoming, outgoing, and missed calls • Phone numbers • Call duration • Timestamps **Extraction Process** • Logical or file system extraction is commonly used • On Android, call logs are stored in **SQLite databases** (e.g., calllog.db) • On iOS, call logs are stored in system databases accessible via backups **Analysis** • SQLite viewers are used to read call history • Deleted call records may be recovered using physical extraction **Forensic Value** • Establishes communication patterns • Helps build timelines **2. Extraction of SMS and MMS Messages** **Data Stored** • SMS content • Sender and receiver numbers • Message timestamps • Delivery status **Extraction Process** • SMS data is extracted via logical or file system acquisition • On Android, messages are stored in **mmssms.db** database • On iOS, SMS/iMessage data is stored in **sms.db** **Recovery of Deleted Messages** • Physical extraction can recover deleted SMS from unallocated memory • Cloud backups (Google Drive, iCloud) may also be analyzed **Forensic Value** • Provides evidence of conversations	**CO5**	**12**

- Useful in fraud, harassment, and conspiracy cases

3. Extraction of Location Data

Sources of Location Data

- GPS coordinates
- Wi-Fi and cell tower logs
- App-based location history (Google Maps, Apple Maps)

Extraction Process

- Location data is obtained from:
  - System location services
  - App databases
  - Cloud backups
- Analysts extract files such as **location cache databases**

Analysis

- Coordinates are mapped using GIS tools
- Timeline analysis correlates movement with events

Forensic Value

- Tracks user movement
- Places suspect at crime scene

4. Extraction of Application Artifacts

What Are App Artifacts?

Artifacts include:

- Chat messages (WhatsApp, Telegram)
- Media files
- Login timestamps
- User preferences and cache data

Extraction Process

- File system extraction is preferred
- App data is stored in:
  - Android: /data/data/app_package_name/
  - iOS: App sandbox directories
- SQLite databases, JSON files, and logs are analyzed

Examples

- WhatsApp chat history (msgstore.db)
- Browser history and cookies
- Banking app transaction logs

Challenges

- Encryption
- Secure containers
- App-level protections

5. Role of Cloud Data in Extraction

Modern apps sync data to the cloud. Analysts may extract:

- iCloud backups
- Google account backups
- App-specific cloud storage

This helps recover **deleted or unavailable local data**.

***** End *****