Lecture #8

Developing a Corporate Mobile Security Strategy

SMA 2025

Developing a Corporate Mobile Security Strategy

From Personal Devices to Enterprise Assets

Today's Agenda

- **Part 1: The Strategic Imperative** - Why every business needs a mobile security strategy.
- **Part 2: The Ownership Dilemma** - BYOD vs. Corporate-Owned vs. Hybrid models.
- **Part 3: The Management Toolkit** - A deep dive into MDM, MAM, and UEM.
- **Part 4: Building the Fortress** - Implementing key technical security controls.
- **Part 5: When Things Go Wrong** - Planning for mobile security incidents.

Recap from Lecture 7

- **The Data Economy:** We explored the world of trackers, ad networks, and data brokers.
- **Advanced Permissions:** We took a deeper look at the iOS and Android permission models.
- **PETs:** We introduced Privacy-Enhancing Technologies.
- **Formal Models:** We demystified k-Anonymity and the mathematical guarantees of Differential Privacy.

Today's Link: While Lecture 7 was about protecting an individual's privacy from the outside world, today is about a company protecting its *own* data on devices used by its employees.

Part 1: The Strategic Imperative

Why Ad-Hoc Mobile Security Fails

The Modern Work Environment

- **Work is no longer a place.** It's an activity. Employees work from home, from coffee shops, from the airport.
- **The phone is the primary computer.** For many employees, especially in sales or field services, their phone or tablet is their main connection to corporate resources.
- **Corporate data is everywhere.** Sensitive emails, presentations, customer data, and financial reports are constantly being synced to and accessed from mobile devices.

What is a Mobile Security Strategy?

A mobile security strategy is a formal, documented plan that defines the **policies, procedures, and controls** for how mobile devices will be used, managed, and secured within an organization.

It's not just a technical document. It answers questions like:

- What devices are allowed?
- Who is responsible for securing them?
- What data can be accessed?
- What happens when a device is lost?

The Cost of Not Having a Strategy

- **Data Breaches:** A lost or compromised employee phone can lead to a massive breach of corporate or customer data.
- **Compliance Violations:** Regulations like GDPR or HIPAA carry heavy fines for failing to protect personal data, regardless of whether it was on a server or a mobile phone.
- **Reputational Damage:** A public breach originating from a mobile device can destroy customer trust.
- **Inconsistent Security:** Without a central policy, some employees might use strong passcodes while others use none, creating weak links in your security chain.

Key Stakeholders

A successful strategy requires buy-in from across the business. It's not just an IT problem.

- **IT Department:** Responsible for implementing and managing the technical tools (UEM).
- **Security Team:** Defines the security policies and manages incident response.
- **Legal & Compliance:** Ensures the strategy complies with regulations like GDPR.
- **Human Resources (HR):** Communicates policies to employees and handles issues related to personal device usage.
- **Finance:** Manages the budget for hardware, software, and potential employee stipends.
- **Executive Leadership:** Provides top-down support and approval for the strategy.

Foundational Step: Mobile Risk Assessment

Before you can write a strategy, you must understand your risks.

- **Identify Assets:** What sensitive data is being accessed by mobile devices? (e.g., customer PII, financial forecasts, source code).
- **Identify Threats:** What are the potential threats to this data? (e.g., device loss/theft, malware, phishing, network attacks).
- **Identify Vulnerabilities:** Where are our weaknesses? (e.g., no MFA, no policy for unpatched devices, users with no passcodes).
- **Analyze Risk:** Determine the likelihood and impact of each threat exploiting a vulnerability. Prioritize the highest risks.

The Mobile Security Policy Document

The strategy is formalized in a policy document. This is the rulebook.

- **Purpose & Scope:** Why does this policy exist and who does it apply to?
- **Policy Statements:** The specific rules. (e.g., "All mobile devices accessing corporate data must be enrolled in the UEM.", "A minimum 6-digit passcode is required.").
- **Roles & Responsibilities:** Who is responsible for what (users, IT, security).
- **Compliance & Enforcement:** What happens if someone violates the policy?
- **Acceptable Use:** What are employees allowed and not allowed to do on managed devices?

Part 2: The Ownership Dilemma

BYOD vs. Corporate-Owned

Model 1: Corporate-Owned (COBO/COPE)

The company buys, owns, and controls the device.

- **COBO (Corporate-Owned, Business Only):** A locked-down device used exclusively for work. Often used in high-security environments.
- **COPE (Corporate-Owned, Personally-Enabled):** A corporate-owned device that the employee is allowed to use for limited personal tasks (e.g., personal email, web browsing).

Corporate-Owned: Pros & Cons

Pros:

- **Maximum Security & Control:** The company can enforce strict policies, block apps, and wipe the device at any time.
- **Simplified Management:** A standardized fleet of devices is easier to manage and secure.
- **Clear Ownership:** No ambiguity about who owns the device and the data on it.

Cons:

- **High Cost:** The company bears the full cost of hardware, service plans, and upgrades.
- **Low Employee Satisfaction:** Employees may not like the device they are given and often resent carrying two phones (one for work, one for personal use).

Model 2: Bring Your Own Device (BYOD)

Employees use their personal smartphones for work.

- The company allows employees to connect their own iPhone or Android device to corporate resources like email and internal apps.
- This is the most popular model in many industries due to its convenience and cost savings.

BYOD: Pros & Cons

Pros:

- **Low Cost:** No hardware costs for the company.
- **High Employee Satisfaction:** Employees get to use the device they prefer.
- **Increased Productivity:** Employees are often more comfortable and efficient on their own devices.

Cons:

- **Major Security & Privacy Challenges:** How do you protect corporate data on a device you don't own? How do you avoid wiping an employee's personal photos?
- **Diverse & Fragmented Fleet:** IT has to support a huge variety of devices, operating systems, and patch levels.
- **Legal Ambiguity:** Who is liable if a data breach occurs on a personal device?

Model 3: Choose Your Own Device (CYOD)

A compromise between COPE and BYOD.

- **How it works:** The company provides a curated list of approved, corporate-purchased devices (e.g., "You can choose the latest iPhone, Samsung Galaxy, or Google Pixel").
- The employee chooses the device they want from the list.
- The device is owned by the company but used as the employee's single, primary phone (similar to COPE).

Comparison of Ownership Models

FEATURE	COPE	CYOD	BYOD
Security Control	High	High	Low-Medium
Hardware Cost	High	High	None
User Satisfaction	Low	Medium	High
IT Complexity	Low	Low-Medium	High

The Employee Privacy Concern

Even on a corporate-owned device, employee privacy is a major concern.

- **What can IT see?** This is the most common question from employees.
- **A UEM can potentially see:** Device model, OS version, installed apps, location, phone number, network information.
- **A UEM generally CANNOT see:** Personal emails, text messages, photos, browsing history, app content.
- **Transparency is Key:** The company's policy must be crystal clear about what is and is not monitored. Building trust with employees is essential for a successful program.

Part 3: The Management Toolkit

MDM, MAM, and UEM

Mobile Device Management (MDM)

- **Focus:** Manages the **entire device** at a hardware and OS level.
- **Capabilities:**
  - Enforce device-wide passcodes and encryption.
  - Configure Wi-Fi, VPN, and email settings.
  - Push or block applications.
- **Remote Wipe:** Completely erase all data on the device if it's lost or stolen.

Common MDM Commands

An MDM server communicates with devices using a specific protocol (like Apple's MDM protocol or Android Enterprise). It can send commands like:

- **DeviceInformation**: Query the device for its model, OS version, serial number, etc.
- **InstallApplication**: Push an app to the device.
- **RemoveApplication**: Uninstall an app.
- **DeviceLock**: Remotely lock the device and display a message.
- **ClearPasscode**: Remove the device's passcode (and then force a new one).
- **EraseDevice**: Trigger a full factory reset (the remote wipe).
- **Restrictions**: Apply or remove device restrictions (e.g., block camera, disable App Store).

Mobile Application Management (MAM)

- **Focus:** Manages only the **corporate applications** on the device, not the device itself.
- **Capabilities:**
  - Creates a secure, encrypted "container" for work apps.
  - Enforce policies *within* the container (e.g., require a PIN to open Outlook).
  - Prevent data leakage (e.g., block copy/paste from a work app to a personal app).
- **Selective Wipe:** Remotely delete only the corporate container and its data, leaving all personal data untouched.

MAM in Action: The Work Container

This is Android's "Work Profile." It creates a completely separate profile on the device for work apps and data. The two profiles are cryptographically isolated. IT can control the work profile, but has no visibility into the personal profile.

MAM App Configuration (Conceptual)

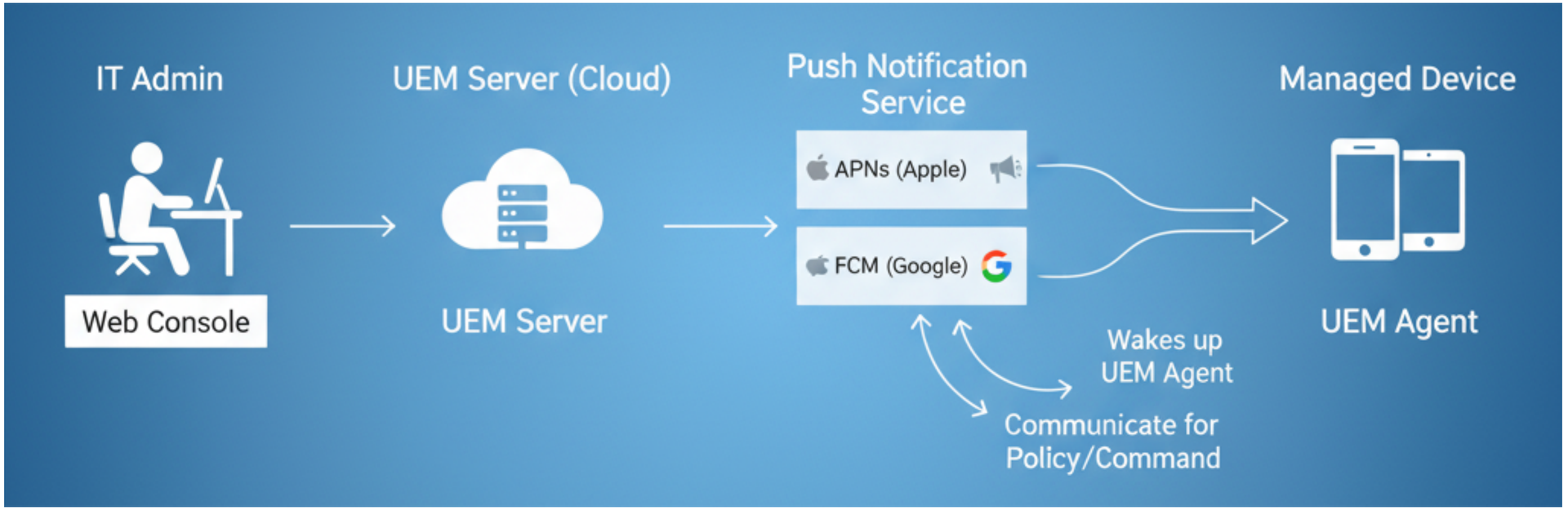
An app can be designed to be "MAM-aware." It can check if it's being managed and apply policies accordingly.

```
// On Android, you can use the RestrictionsManager to check for managed configurations

val restrictionsManager = context.getSystemService(Context.RESTRICTIONS_SERVICE) as RestrictionsManager

// Get the configuration set by the UEM/MAM server
val appRestrictions = restrictionsManager.applicationRestrictions

// Apply policies based on the configuration
val allowScreenshots = appRestrictions.getBoolean("key_allow_screenshots", true)
if (!allowScreenshots) {
    window.setFlags(WindowManager.LayoutParams.FLAG_SECURE, WindowManager.LayoutParams.FLAG_SECURE)
}

val serverUrl = appRestrictions.getString("key_server_url")
if (serverUrl != null) {
    // Automatically configure the app with the correct server URL
}
```

Unified Endpoint Management (UEM)

- **The Evolution:** UEM is the modern term for platforms that combine MDM, MAM, and management for other devices (like desktops, laptops, and IoT devices) into a single console.
- **Key Players:**
  - Microsoft Intune
  - VMware Workspace ONE
  - Jamf (for Apple devices)
  - MobileIron (now part of Ivanti)

UEM Architecture



The Enrollment Process

How does a device get into a managed state in the first place?

- **Manual Enrollment (for BYOD):** The user downloads the UEM agent app (e.g., the Intune Company Portal), signs in, and follows on-screen steps to install the management profile.
- **Automated Device Enrollment (for Corporate-Owned):**
- **Apple Business Manager:** Allows a company to associate device serial numbers with their UEM server before they are even unboxed. When the user turns on the new iPhone, it automatically enrolls itself during the setup wizard.
- **Android Zero-Touch Enrollment:** The equivalent for Android devices.

Part 4: Building the Fortress

Implementing Key Technical Security Controls

Control 1: Access Control

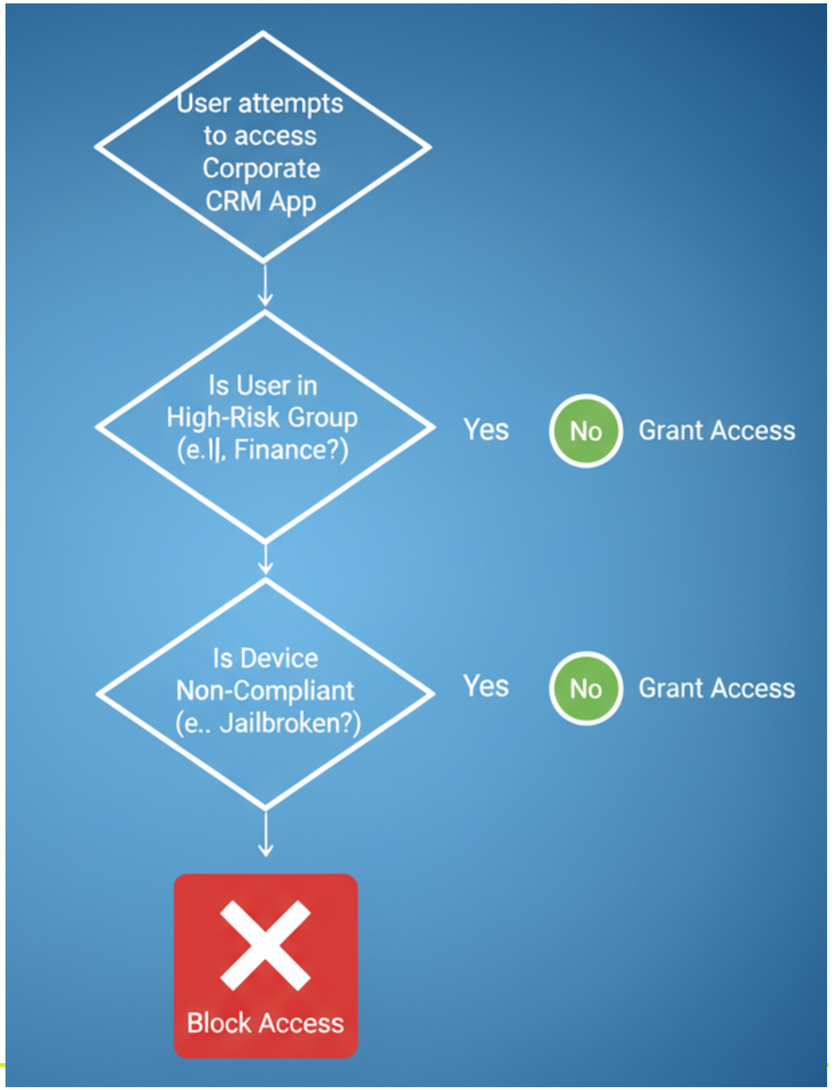
- **Enforce Strong Authentication:**
  - Require a device-level passcode or biometric lock on all devices that access corporate data.
  - Use your UEM tool to set minimum passcode complexity and an auto-lock timer.
  - All access to corporate resources (email, VPN, cloud apps) must be protected by MFA, as we discussed in Lecture 6.

Advanced Access Control: Conditional Access

This is a key feature of modern identity and UEM platforms.

Concept: Access is not a simple yes/no decision. It is granted based on a set of conditions or signals.

Policy Example: *IF* a user is trying to access the corporate CRM app, *AND* the user is in a high-risk group (e.g., Finance), *AND* their device is marked as non-compliant by the UEM (e.g., it's jailbroken), *THEN* block access.



Control 2: Data Protection

- **Enforce Encryption:**
  - Modern mobile devices are encrypted by default when a passcode is set. Your UEM policy should verify and enforce this.
  - Require all connections to corporate resources to use strong TLS (TLS 1.2 or 1.3).
  - Use a VPN for access to internal network resources.
  - For custom-built apps, ensure developers are using secure storage APIs (Keychain/ EncryptedSharedPreferences) to store sensitive data, as discussed in Lecture 2.

Advanced Data Protection: Per-App VPN

Instead of a device-wide VPN that routes all traffic through the corporate network, a Per-App VPN is more granular.

- **How it works:** The UEM configures the device so that only traffic from specific, managed applications goes through the VPN.
- Traffic from the user's personal apps (e.g., Netflix, Spotify) goes directly to the internet.
- **Benefits:**
  - **Improves Privacy:** The company doesn't see the user's personal internet traffic.
  - **Improves Performance:** Reduces the load on the corporate VPN infrastructure.
  - **Saves Battery:** Less network overhead on the device.

Control 3: Data Leakage Prevention (DLP)

This is a key feature of MAM.

- **Goal:** Prevent users from intentionally or accidentally moving corporate data from managed apps to unmanaged apps.
- **Controls:**
  - **Block Copy/Paste:** Prevent copying text from a work email in Outlook and pasting it into a personal WhatsApp message.
  - **Control "Open In":** Prevent a user from opening a sensitive PDF attachment from their work email in an unmanaged personal app.
  - **Block Screenshots/Screen Recording:** Disable the ability to take screenshots of managed applications.

DLP in Code (Android)

A developer can programmatically block screenshots in their app.

When this flag is set, the Android OS prevents screenshots, screen recording, and mirroring of this app's window.

```
// In your Activity's onCreate method

if (isProductionBuild()) { // Only do this in production
    window.setFlags(
        WindowManager.LayoutParams.FLAG_SECURE,
        WindowManager.LayoutParams.FLAG_SECURE
    )
}
```

DLP in Code (iOS)

iOS does not have a direct equivalent to *FLAG_SECURE*, but developers can detect when a screenshot is taken and react.

```
// In your ViewController

override func viewDidLoad() {
    super.viewDidLoad()

    // Add an observer for the screenshot notification
    NotificationCenter.default.addObserver(
        self,
        selector: #selector(didTakeScreenshot),
        name: UIApplication.userDidTakeScreenshotNotification,
        object: nil
    )
}

@objc func didTakeScreenshot() {
    // The user took a screenshot.
    // You can't prevent it, but you can react.
    print("Screenshot detected!")
    // For example, you could blur the screen, log the event,
    // or remind the user not to share sensitive information.
    showAlert(title: "Security Warning", message: "Please do not share screenshots of this sensitive data.")
}
```

Control 4: Application Management

- **Private App Store:**
  - Use your UEM platform to create an enterprise app store to securely distribute in-house and approved third-party apps.
- **Whitelisting (most secure):** Only allow apps from the approved list to be installed.
- **Blacklisting (more flexible):** Block specific, known-bad or high-risk apps (e.g., apps known to contain malware, or apps from untrusted developers).

Advanced App Management: App Wrapping

What if you need to apply MAM policies to an app that you didn't build, and it isn't MAM-aware?

- **App Wrapping** is a technology that takes a compiled application (*.apk* or *.ipa*) and injects the MAM management code into it *without* needing the source code.
- **Process:**
  - Upload the compiled app to the App Wrapping tool.
  - The tool decompiles the app, adds the MAM SDK and policy hooks, and then recompiles it.
  - The new, "wrapped" app can now be managed by the UEM.

Control 5: Threat Detection & Response

- **Mobile Threat Defense (MTD):**
  - These are apps that act like an antivirus for your phone. They are often integrated with UEM platforms.
  - They scan for malware, detect phishing attacks, and identify risky device configurations (like being jailbroken or rooted).
  - The UEM can be configured to automatically take action if the MTD tool detects a threat.
  - Example: If malware is detected on a device, the UEM can automatically block its access to corporate email and VPN until the threat is removed.

MTD Threat Vectors

Mobile Threat Defense tools look for threats at three main levels:

- **Device-Level Threats:**
  - Is the device jailbroken or rooted?
  - Is the OS outdated and vulnerable?
  - Is a malicious configuration profile installed?
  - Is the device connected to a malicious Wi-Fi hotspot?
  - Is it experiencing a Man-in-the-Middle (MitM) attack?
  - Is it communicating with a known malware command-and-control server?
  - Is there known malware installed?
  - Is an app exhibiting suspicious behavior (e.g., a sideloaded app asking for Accessibility Service access)?

Control 6: Patch Management

- **The Problem:** Mobile OS vulnerabilities are discovered all the time. Unpatched devices are a massive security risk.
- **The Policy:** Your strategy must define the minimum acceptable OS version for devices to connect to the network.
- **The Enforcement:**
  - Your UEM can check the OS version of a device during every connection attempt.
  - If the device is running an outdated, vulnerable OS, access can be blocked.
  - The user can be redirected to a page with instructions on how to update their device.

Part 5: When Things Go Wrong

Mobile Incident Response

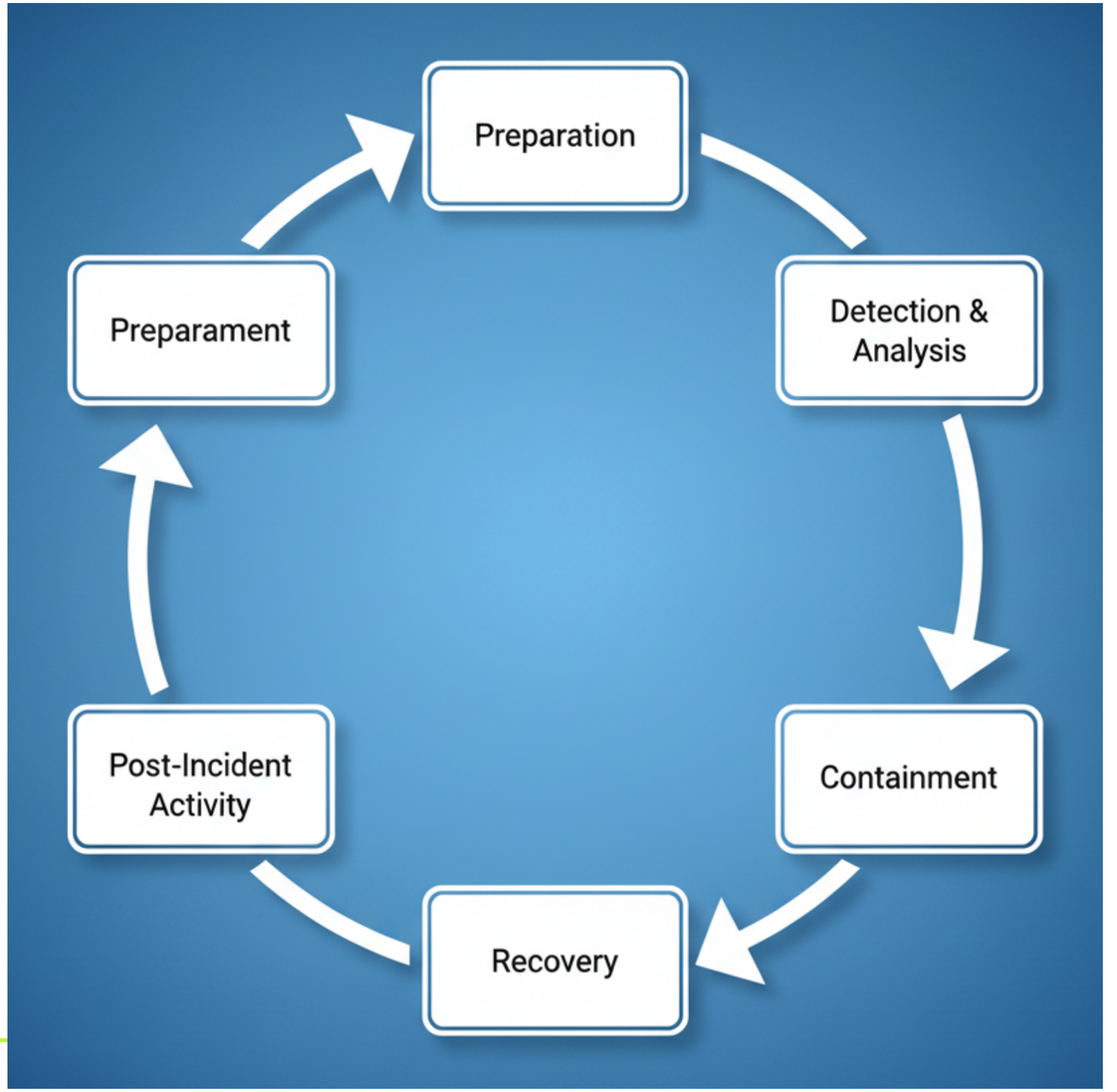
What is Incident Response?

Incident Response (IR) is the process of responding to and managing the aftermath of a security breach or attack.

The goal is to **contain the damage, eradicate the threat, and recover normal operations** as quickly as possible.

The Six Phases of Incident Response (NIST)

- **Preparation:** Having a plan and the right tools in place *before* an incident.
- **Detection & Analysis:** Discovering that an incident has occurred and determining its scope.
- **Containment:** Stopping the incident from causing further damage.
- **Eradication:** Removing the root cause of the incident.
- **Recovery:** Restoring systems to normal operation.
- **Post-Incident Activity:** Learning from the incident to improve future defenses.



Phase 1: Preparation

This is the most critical phase.

- **Develop a Mobile IR Plan:** Who do you call when a phone is lost? What are the steps? This must be written down in a playbook.
- **Train Your Users:** Employees must know how to report a lost or stolen device immediately. Every minute counts. They need a single, 24/7 phone number or portal to use.
- **Train Your IT/Security Team:** The help desk and security team need to be drilled on the playbook. They need to know how to perform a remote wipe, how to investigate a compromised device, etc.
- **Ensure You Have the Right Tools:** You can't perform a remote wipe if you haven't deployed a UEM. You can't detect malware without an MTD solution.

Mobile IR Playbook Example: Lost Device

A simplified playbook for the IT Help Desk:

- **Receive Report:** User reports a lost device via phone call or portal.
- **Verify User Identity:** Verify the user's identity using a pre-defined procedure (e.g., security questions, call-back to a registered number).
- **Locate Device:** Attempt to locate the device using the UEM's GPS tracking feature.
- **Initiate Remote Lock:** Immediately execute the *DeviceLock* command from the UEM console. Display a message on the screen (e.g., "This phone is lost. Please call XXX-XXX-XXXX").
- **Assess Risk:** Based on the user, data on the device, and last known location, determine if a remote wipe is necessary.
- **Execute Remote Wipe:** If deemed necessary, execute the *EraseDevice* (for COPE) or selective wipe (for BYOD) command.
- **Revoke Access:** Concurrently, revoke the user's session tokens and force a password reset for their corporate accounts.
- **Document:** Log all actions taken in the incident tracking system.

Phase 2: Detection & Analysis

- **How are mobile incidents detected?**
- **User Report:** The most common source. "I lost my phone!" or "My phone is acting weird."
- **Automated Alert:** An alert from the UEM or MTD tool (e.g., "Malware detected on Device XYZ").
- **Log Analysis:**
  - A security analyst notices suspicious activity from a device in the network logs (e.g., connecting to a malicious IP).
  - Who is the user? What is their role?
  - What data is on the device?
  - What is the nature of the threat?
  - Is this an isolated incident or part of a larger campaign?

Phase 3: Containment

Goal: Stop the bleeding. Prevent the incident from spreading or causing more damage.

- **Short-Term Containment:**
  - **Quarantine the device:** Use the UEM to block all network access.
  - **Isolate the user account:** Temporarily disable the user's corporate accounts.
  - **Remote Wipe:** The ultimate containment strategy for a lost or unrecoverable device.

Phase 4: Eradication

Goal: Remove the root cause of the threat.

- **For a lost device:** The remote wipe eradicates the threat by destroying the data.
- **For a malware infection:**
  - The MTD tool may be able to automatically remove the malware.
  - The user may be instructed to uninstall the malicious app.
  - In severe cases, the only way to be 100% sure the threat is gone is to perform a full factory reset of the device.

Phase 5: Recovery

Goal: Restore normal business operations.

- **Provision a new device:** For a lost or wiped device, get a new, clean, managed device into the employee's hands.
- **Restore data:** Restore corporate data (like email and contacts) from server-side backups.
- **Re-enable access:** Once the device is clean and secure, remove it from quarantine and restore its access to corporate resources.
- **Monitor closely:** For a period after the incident, the device and user account should be monitored more closely for any signs of unusual activity.

Phase 6: Post-Incident Activity (Lessons Learned)

After every incident, the team should conduct a post-mortem review.

- What happened, and why? What was the root cause?
- How did our security controls perform? Did the MTD detect the malware? Did the DLP policy prevent data exfiltration?
- How did our people and processes perform? Was the incident reported quickly? Did the help desk follow the playbook?
- What could we have done better?
- Do we need to update our policies, tools, or user training?

An incident is a learning opportunity. The goal is to ensure the same type of incident doesn't happen again.

The Forensic Challenge

A remote wipe is great for containment, but it destroys all evidence of what happened on the device.

- **The Dilemma:** In the case of a sophisticated attack or a malicious insider, you may want to analyze the device to understand the attacker's methods. But your first instinct is to wipe it.
- **The Solution:** The IR plan must have a process for escalating certain incidents to a digital forensics team *before* the wipe command is issued.
- This requires a difficult risk decision: Is the need to preserve evidence greater than the immediate need to destroy the data?

Appendix: Developer's Corner

From Strategy to Code: Enforcing Corporate Policies in Apps

In this appendix, we connect the high-level strategy to concrete implementation details developers can use when building corporate mobile apps.

Android: Enforcing Screen Lock & Encryption (Conceptual)

Goal: Ensure corporate apps only run on devices that meet minimum security posture.

Key Idea: Use the Android Enterprise / Device Policy APIs (for device-owner/profile-owner apps) to enforce:

- A screen lock (PIN/password/pattern).
- Device encryption.

```
// In a device/profile owner app managed by UEM
val dpm = getSystemService(DevicePolicyManager::class.java)
val admin = ComponentName(this, MyDeviceAdminReceiver::class.java)

// 1) Require a screen lock
//   e.g., minimum 6 digits, no simple (1234, 0000) patterns
val quality = DevicePolicyManager.PASSWORD_QUALITY_NUMERIC

dpm.setPasswordQuality(admin, quality)
dpm.setPasswordMinimumLength(admin, 6)

// 2) Require device encryption (where supported)
if (dpm.getStorageEncryptionStatus() ==
        DevicePolicyManager.ENCRYPTION_STATUS_INACTIVE) {
    dpm.setStorageEncryption(admin, true)
}
```

iOS: Passcode Policy via Configuration Profile

On iOS, device-wide security requirements (like passcode strength) are not enforced by your app directly, but by **MDM configuration profiles**.

Example: Passcode Policy Payload (Excerpt)

```
<!-- PayloadType: com.apple.mobiledevice.passwordpolicy -->
<dict>
  <key>PayloadType</key>
  <string>com.apple.mobiledevice.passwordpolicy</string>
  <key>PayloadIdentifier</key>
  <string>com.example.passcodepolicy</string>
  <key>PayloadUUID</key>
  <string>ABCD-1234-...</string>

  <!-- Require a passcode and minimum length of 6 -->
  <key>maxPINAgeInDays</key>
  <integer>90</integer>
  <key>minLength</key>
  <integer>6</integer>
  <key>forcePIN</key>
  <true/>

  <!-- Wipe device after 10 failed attempts -->
  <key>maxFailedAttempts</key>
  <integer>10</integer>
</dict>
```

Android: Blocking Camera in the Work Profile

Scenario: Corporate policy says "No camera in the work profile" (e.g., in sensitive environments).

Approach: Use the profile-owner's *DevicePolicyManager* to disable the camera.

```
val dpm = getSystemService(DevicePolicyManager::class.java)
val admin = ComponentName(this, MyDeviceAdminReceiver::class.java)

// Disable camera in the managed (work) profile only
val disable = true

if (dpm != null) {
    dpm.setCameraDisabled(admin, disable)
}
```

iOS: Managed Open-In and Data Flow

On iOS, data leakage between managed (corporate) and unmanaged (personal) apps is controlled via the *com.apple.applicationaccess* payload.

Example: Block Data from Managed → Unmanaged Apps

```
<!-- PayloadType: com.apple.applicationaccess -->
<dict>
  <key>PayloadType</key>
  <string>com.apple.applicationaccess</string>
  <key>PayloadIdentifier</key>
  <string>com.example.appaccess</string>

  <!-- Block sharing from managed apps to unmanaged apps -->
  <key>allowOpenFromManagedToUnmanaged</key>
  <false/>

  <!-- But allow data from unmanaged → managed (e.g., attach a personal PDF into Outlook) -->
  <key>allowOpenFromUnmanagedToManaged</key>
  <true/>
</dict>
```

Android: Per-App VPN Policy (Conceptual JSON)

Per-App VPN is often configured in UEM as a JSON policy that links specific apps to a VPN configuration.

```
{
 "vpnConfigId": "corp-vpn-1",
 "packageNames": [
  "com.example.corp.crm",
  "com.example.corp.files"
 ],
 "onWifiOnly": true
}
```

Mini-Lab 1 (Android): Secure Corporate Notes App

Goal: Build a simple "Corporate Notes" app that:

- Locks its UI behind the device's biometric/PIN.
- Stores notes using encrypted preferences.

```
class SecureActivity : AppCompatActivity() {

    private lateinit var biometricPrompt: BiometricPrompt
    private lateinit var promptInfo: BiometricPrompt.PromptInfo

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        promptInfo = BiometricPrompt.PromptInfo.Builder()
            .setTitle("Unlock Corporate Notes")
            .setSubtitle("Authenticate to access work data")
            .setAllowedAuthenticators(
                BiometricManager.Authenticators.BIOMETRIC_STRONG or
                    BiometricManager.Authenticators.DEVICE_CREDENTIAL
            )
            .build()

        biometricPrompt = BiometricPrompt(
            this,
            ContextCompat.getMainExecutor(this),
            object : BiometricPrompt.AuthenticationCallback() {
                override fun onAuthenticationSucceeded(result: BiometricPrompt.AuthenticationResult) {
                    // Show the notes UI only after success
                    setContentView(R.layout.activity_secure_notes)
                }
```

Goal: Build a simple "Corporate Notes" app that:

- Locks its UI behind the device's biometric/PIN.
- Stores notes using encrypted preferences.

```
    private lateinit var biometricPrompt: BiometricPrompt
    private lateinit var promptInfo: BiometricPrompt.PromptInfo

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        promptInfo = BiometricPrompt.PromptInfo.Builder()
            .setTitle("Unlock Corporate Notes")
            .setSubtitle("Authenticate to access work data")
            .setAllowedAuthenticators(
                BiometricManager.Authenticators.BIOMETRIC_STRONG or
                    BiometricManager.Authenticators.DEVICE_CREDENTIAL
            )
            .build()

        biometricPrompt = BiometricPrompt(
            this,
            ContextCompat.getMainExecutor(this),
            object : BiometricPrompt.AuthenticationCallback() {
                override fun onAuthenticationSucceeded(result: BiometricPrompt.AuthenticationResult) {
                    // Show the notes UI only after success
                    setContentView(R.layout.activity_secure_notes)
                }
            }
        )

        biometricPrompt.authenticate(promptInfo)
    }
}
```

Mini-Lab 1 (iOS): Secure Corporate Notes App

Goal: Implement the same idea for iOS using *LocalAuthentication*.

```
import LocalAuthentication
import UIKit

class SecureViewController: UIViewController {

    override func viewDidAppear(_ animated: Bool) {
        super.viewDidAppear(animated)
        authenticateUser()
    }

    private func authenticateUser() {
        let context = LAContext()
        let reason = "Authenticate to access corporate notes."

        context.evaluatePolicy(.deviceOwnerAuthentication,
                               localizedReason: reason) { success, error in
            DispatchQueue.main.async {
                if success {
                    // Show secure content
                    self.showNotesScreen()
                } else {
                    // Dismiss or show an error
```

```
    override func viewDidAppear(_ animated: Bool) {
        super.viewDidAppear(animated)
        authenticateUser()
    }

    private func authenticateUser() {
        let context = LAContext()
        let reason = "Authenticate to access corporate notes."

        context.evaluatePolicy(.deviceOwnerAuthentication,
                               localizedReason: reason) { success, error in
            DispatchQueue.main.async {
                if success {
                    // Show secure content
                    self.showNotesScreen()
                } else {
                    // Dismiss or show an error
                    self.dismiss(animated: true)
                }
            }
        }
    }

    private func showNotesScreen() {
        // Present your notes UI here
    }
}
```

Mini-Lab 2: Secure Token Storage (Android)

Goal: Store an access token for a corporate backend in a secure way.

```
import androidx.security.crypto.EncryptedSharedPreferences
import androidx.security.crypto.MasterKey

fun createSecurePrefs(context: Context): SharedPreferences {
    val masterKey = MasterKey.Builder(context)
        .setKeyScheme(MasterKey.KeyScheme.AES256_GCM)
        .build()

    return EncryptedSharedPreferences.create(
        context,
        "corp_prefs",
        masterKey,
        EncryptedSharedPreferences.PrefKeyEncryptionScheme.AES256_SIV,
        EncryptedSharedPreferences.PrefValueEncryptionScheme.AES256_GCM
    )
}

fun saveToken(prefs: SharedPreferences, token: String) {
    prefs.edit().putString("access_token", token).apply()
}
```

Mini-Lab 2: Secure Token Storage (iOS)

Goal: Store the same kind of access token securely on iOS using the Keychain.

```
import Security

func saveToken(token: String) {
    let data = token.data(using: .utf8)!

    let query: [String: Any] = [
        kSecClass as String: kSecClassGenericPassword,
        kSecAttrAccount as String: "corp.access.token",
        kSecValueData as String: data,
        kSecAttrAccessible as String: kSecAttrAccessibleWhenUnlockedThisDeviceOnly
    ]

    // Remove any existing item
    SecItemDelete(query as CFDictionary)

    // Add the new token
    SecItemAdd(query as CFDictionary, nil)
}
```

Mini-Lab 3: Using Managed Config (Android)

Goal: Let the UEM configure your app (e.g., server URL, feature flags) without hardcoding values.

```
val restrictionsManager =
    getSystemService(Context.RESTRICTIONS_SERVICE) as RestrictionsManager

val appRestrictions = restrictionsManager.applicationRestrictions

val serverUrl = appRestrictions.getString("corp_server_url")
val screenshotsAllowed =
    appRestrictions.getBoolean("corp_allow_screenshots", false)

if (!screenshotsAllowed) {
    window.setFlags(
        WindowManager.LayoutParams.FLAG_SECURE,
        WindowManager.LayoutParams.FLAG_SECURE
    )
}
```

Mini-Lab 3: Feature Flags via MDM (iOS)

Goal: Achieve a similar effect on iOS using the *com.apple.configuration.managed* dictionary.

```
import UIKit

func loadManagedConfig() -> [String: Any] {
    let defaults = UserDefaults.standard
    let key = "com.apple.configuration.managed"
    return defaults.dictionary(forKey: key) ?? [:]
}

func applySecuritySettings() {
    let config = loadManagedConfig()
    if let allowScreenshots = config["corp_allow_screenshots"] as? Bool,
       allowScreenshots == false {
        // iOS does not let you fully block screenshots,
        // but you can react (e.g., blur sensitive views when app goes to background)
    }
}
```

Connecting the Dots

How the Developer's Corner Relates to the Strategy

- Strategy defines **what** must be protected (data, devices, identities).
- UEM/MDM defines **where** and **when** policies apply.
- Application code defines **how** those policies are enforced in practice.

Examples:

- A policy says "No screenshots of corporate data" ->’ App uses *FLAG_SECURE* (Android) or reacts to screenshot notifications (iOS).
- A policy says "Use strong device authentication" ->’ Device policy enforces passcode / biometrics, app gates access using *BiometricPrompt* / *LocalAuthentication*.

Project Ideas

Consider integrating one or more of these ideas into your semester project:

- **Corporate Mode:** A setting that turns on extra protections (biometric gate, DLP, stricter timeouts) when the app is used with a corporate backend.
- **UEM-Aware Configuration:** Reading server URLs, feature flags, or log levels from managed config instead of hardcoding them.
- **Secure Offline Cache:** Combining local database encryption with secure key storage for offline access to corporate data.
- **Risk-Based UX:** Making certain operations (e.g., exporting a report) require re-authentication.

Key Takeaways (1/2)

- **Strategy is Essential:** You need a formal, documented plan to manage the risks of mobile devices in the enterprise. This plan should be based on a thorough risk assessment.
- **Choose Your Model:** The BYOD vs. Corporate-Owned decision is the foundational policy choice that dictates your strategy. Most companies use a hybrid model.
- **UEM is the Core Tool:** Modern UEM platforms (combining MDM and MAM) are essential for enrolling devices, enforcing policy, and managing a mobile fleet at scale.

Key Takeaways (2/2)

- **Defense in Depth:** A robust strategy involves multiple layers of technical controls, including access control (MFA, Conditional Access), data protection (Encryption, DLP), application management, and threat detection (MTD).
- **Plan for Failure:** A well-rehearsed Incident Response plan, based on the six phases of IR, is just as important as your preventative controls.

Q&A

Questions?

-