Lecture #13

Hands-on Mobile & IoT Penetration Testing

SMA 2025

Title Slide

Hands-on Mobile & IoT Penetration Testing

From Lab Setup to Exploitation

Today's Agenda

- **Part 1: The Pentest Lab** - Setting up your attack environment.
- **Part 2: Static Analysis (SAST)** - Finding bugs without running the app.
- **Part 3: Dynamic Analysis (DAST)** - Manipulating apps at runtime with Frida.
- **Part 4: Network Interception** - Breaking TLS with Burp Suite.
- **Part 5: IoT Security Testing** - Sniffing BLE and MQTT.
- **Part 6: Reporting** - How to write a vulnerability report.

Disclaimer

Ethical Hacking Only

- The tools and techniques discussed today are for **educational purposes only**.
- Only test applications you own or have explicit written permission to test.
- Testing third-party apps without permission is illegal and can lead to prosecution.

Part 1: The Pentest Lab

Building Your Arsenal

Emulator vs. Physical Device

- **Emulators (Genymotion, Android Studio AVD):**
- **Pros:** Free, easy to root, snapshot capability (save state).
- **Cons:** No Bluetooth, no NFC, some ARM libraries won't run (x86 architecture).
- **Pros:** Real hardware (Bluetooth/NFC works), 100% app compatibility.
- **Cons:** Risk of bricking, expensive.

The "Root" Requirement

- **Why Root/Jailbreak?**
- To access the app's private data (*/data/data/com.app*).
- To hook system functions (Frida).
- To bypass SSL Pinning.
- To intercept traffic.

Essential Tools Installation

- **ADB (Android Debug Bridge):** The command line tool to talk to Android.
- **Frida:** The dynamic instrumentation toolkit.

Part 2: Static Analysis (SAST)

Reading the Blueprint

MobSF: The Swiss Army Knife

- **What it is:** An automated tool that decompiles the app and scans for vulnerabilities.
- **How to use:**
- Drag and drop your APK or IPA file into the web interface.
- Wait for the scan to finish.
- Read the report.
- Hardcoded API keys.
- Insecure permissions.
- Weak crypto configurations.
- Firebase database misconfigurations.

Manual Reversing with JADX

- **JADX:** A GUI tool to decompile Android APKs to Java.
- **Why use it?** MobSF gives you a high-level overview, but JADX lets you read the logic.
- **Workflow:**
- Open APK in JADX-GUI.
- Search for strings like "password", "api_key", "token".
- Read the *LoginActivity* to see how authentication is handled.

iOS Reversing: Ghidra & Hopper

- **The Challenge:** iOS apps are compiled to machine code (ARM64), not bytecode. They are much harder to read.
- **Tools:**
  - **Ghidra:** NSA's open-source reverse engineering suite.
  - **Hopper Disassembler:** A paid but user-friendly Mac tool.
- Symbol names (function names) that give away logic.
- String constants.

Part 3: Dynamic Analysis (DAST)

Manipulating Reality with Frida

What is Frida?

- **Concept:** A dynamic instrumentation toolkit. It lets you inject JavaScript into a running app to modify its behavior.
- **Capabilities:**
  - Read/Write memory.
  - Intercept function calls.
  - Change return values (e.g., make *isAdmin()* return *true*).
  - Bypass SSL Pinning.

Setting up Frida

- **Install Python bindings:** *pip install frida-tools*
- **Download Frida Server:** Get the binary matching your device architecture (arm64) from GitHub.
- **Push to Device:**adb push frida-server /data/local/tmp/ adb shell "chmod 755 /data/local/tmp/frida-server" adb shell "/data/local/tmp/frida-server &"
- **Verify:** *frida-ps -U* (Lists running processes on USB device).

Frida Scripting: The Basics

Goal: Bypass a function *checkPin(String pin)* that returns a boolean.

```
Java.perform(function() {
    // 1. Get a reference to the class
    var MainActivity = Java.use("com.example.app.MainActivity");

    // 2. Hook the method
    MainActivity.checkPin.implementation = function(pin) {
        console.log("Intercepted PIN check for: " + pin);

        // 3. Force return true
        return true;
    };
});
```

Objection: Frida without Scripting

- **What it is:** A command-line tool that automates common Frida tasks.
- **Key Commands:**
  - **android sslpinning disable**: Automatically bypasses pinning.
  - **android root disable**: Hides root from the app.
  - **ios keychain dump**: Dumps the iOS Keychain.
  - **memory search "password"**: Searches RAM for strings.

Part 4: Network Interception

Breaking TLS with Burp Suite

The Setup: Proxying Traffic

- **Burp Suite:** Start the Proxy listener on *All Interfaces*.
- **Phone:** Go to Wi-Fi Settings -> Advanced -> Proxy -> Manual.
- Host: Your Laptop's IP.
- Port: 8080.
- Visit *http://burp* on the phone browser.
- Download the CA Certificate.
- Install it as a "Trusted Root" in Android/iOS settings.

Bypassing SSL Pinning

- **The Problem:** Even with the cert installed, secure apps will reject it because they "pin" the real server cert.
- **The Fix:** Use Frida/Objection to disable the pinning check logic in the app.
- Command: *objection --gadget com.app.name explore --startup-command "android sslpinning disable"*

Analyzing API Traffic

- **What to look for:**
  - **IDOR (Insecure Direct Object Reference):** Change *user_id=100* to *user_id=101*. Can you see someone else's data?
  - **Excessive Data Exposure:** Does the API return the full user object (including password hash) even if the UI only shows the name?
  - **Broken Auth:** Can you reuse an old token?

Part 5: IoT Security Testing

Hacking the Hardware Connection

Bluetooth Low Energy (BLE) Basics

- **GATT (Generic Attribute Profile):** The architecture of BLE data.
- **Services:** Collections of characteristics (e.g., "Heart Rate Service").
- **Characteristics:** The actual data points (e.g., "Heart Rate Measurement").
- Can be Read, Written, or Notified (Updates).

Tools for BLE Hacking

- **nRF Connect (Mobile App):** The best tool for exploring BLE devices. It lets you scan, connect, and read/write characteristics.
- **Bettercap (Linux):** A powerful tool for BLE sniffing and spoofing.
- **Ubertooth One:** Hardware sniffer for capturing BLE packets in the air.

BLE Attack: Replay

- **Scenario:** A smart lock opens when you press "Unlock" in the app.
- **Attack:**
  - Capture the "Unlock" packet using a sniffer (Ubertooth).
  - Wait for the owner to leave.
  - Replay the exact same packet.

MQTT Hacking

- **MQTT:** The messaging protocol for IoT (Publish/Subscribe).
- **The Vulnerability:** Open brokers.
  - Many developers leave the MQTT broker exposed to the internet with no password.
  - The # wildcard subscribes to EVERYTHING.

Part 6: Reporting

Turning Hacks into Fixes

Elements of a Good Report

- **Title:** Concise description (e.g., "Hardcoded AWS Credentials in strings.xml").
- **Severity:** Critical, High, Medium, Low (use CVSS score).
- **Description:** What is the bug?
- **Steps to Reproduce:** Step-by-step guide so the developer can see it.
- **Impact:** What can an attacker do? (e.g., "Steal all user data").
- **Remediation:** How to fix it (e.g., "Use EncryptedSharedPreferences").

CVSS (Common Vulnerability Scoring System)

- **The Industry Standard:** A calculator to determine severity.
- **Metrics:**
  - **Attack Vector:** Network vs. Physical.
  - **Complexity:** Low vs. High.
  - **Privileges Required:** None vs. Admin.
  - **Impact:** Confidentiality, Integrity, Availability.

Part 7: Final Project Workshop Guidelines

Applying This to Your Project

Project Requirements

- **The Goal:** Build a secure mobile app OR analyze an insecure one.
- **Option A (Builder):** Build an app that implements:
  - Secure Storage (EncryptedSharedPreferences).
  - Certificate Pinning.
  - Biometric Auth.
  - Find 5 vulnerabilities.
  - Demonstrate them with Frida/Burp.
  - Propose fixes.

Peer Review Session

- **Pair Up:** Find a partner.
- **Exchange APKs:** If you are a builder, give your APK to a breaker.
- **The Challenge:** Can the breaker bypass your security?
- Can they find the API key?
- Can they bypass your pinning?

Lab Exercise 1: Setting up MobSF

[Activity]

- Pull the MobSF Docker image.
- Run it.
- Download the "InsecureBankv2.apk" (provided in LMS).
- Upload it to MobSF.
- **Question:** What is the package name? What permissions are dangerous?

Lab Exercise 2: JADX Hunting

[Activity]

- Open "InsecureBankv2.apk" in JADX.
- Search for "LoginActivity".
- Find the *login()* function.
- **Question:** Is the username/password hardcoded? Or where is it sent?

Lab Exercise 3: Frida Hooking

[Activity]

- Install the "Root Detection" demo app.
- Run it. It says "Device is Rooted!" and closes.
- Write a Frida script to hook *isRooted()* and return *false*.
- Run the script: *frida -U -f com.demo.root -l script.js*.
- **Result:** The app should open normally.

Lab Exercise 4: Burp Interception

[Activity]

- Configure phone proxy to your laptop.
- Open the browser on the phone and go to *google.com*.
- Check Burp Suite "Proxy" tab.
- **Question:** Do you see the request? If not, did you install the CA cert?

Common Pitfalls in Testing

- **Emulator Issues:** Some apps crash on x86 emulators. Use an ARM translation tool or a physical device.
- **Network Isolation:** Ensure your laptop and phone are on the same Wi-Fi network for Burp to work.
- **Certificate Trust:** On Android 7+, user certs are not trusted by apps. You must move the Burp cert to the System store (requires root).

Advanced Frida: Native Hooking

- **Java vs. Native:** We hooked Java methods. But what if the logic is in C++ (*.so* file)?
- **Interceptor:** Frida can hook native functions too.
- **Code:**

```
Interceptor.attach(Module.findExportByName("libnative.so", "check_license"), {
  onLeave: function(retval) {
    retval.replace(1); // Return True
  }
});
```

Automating with Python

- You can drive Frida from Python to automate attacks.
- **Example:** Brute-forcing a PIN.
- Python script calls *checkPin("0000")*.
- Checks result.
- Calls *checkPin("0001")*.
- Repeats until success.

IoT Lab: Simulating a BLE Device

- **Tool:** LightBlue (iOS/Mac) or nRF Connect (Android).
- **Action:** Create a "Virtual Peripheral".
- Add a service "Smart Lock".
- Add a characteristic "Lock State".

IoT Lab: MQTT Sniffing

- **Target:** *test.mosquitto.org* (Public test broker).
- **Action:** Subscribe to # (everything).
- **Observation:** Watch the chaos. You will see people's temperature sensors, lights, and test messages from all over the world.
- **Warning:** Do not interact with anything. Just look.

Writing the Report: The Executive Summary

- **Audience:** Management (Non-technical).
- **Content:**
- "We tested App X."
- "We found 3 Critical issues."
- "The app is currently unsafe for production."

Writing the Report: Technical Details

- **Audience:** Developers.
- **Content:**
- HTTP Request/Response logs.
- Screenshots of code.
- Video of the exploit.

Remediation: Defense in Depth

- **Don't just fix the bug.** Fix the process.
- **Example:** If you found a hardcoded key:
- Remove the key.
- Rotate the key (invalidate the old one).
- Add a pre-commit hook to scan for keys so it doesn't happen again.

Tools for iOS Jailbreaking

- **Checkra1n:** Hardware exploit (Bootrom) for iPhone X and older. Unpatchable.
- **Palera1n:** For newer iOS versions.
- **Cydia / Sileo:** The "App Store" for jailbroken apps (install Frida, SSH, Filza).

iOS: Bypassing Jailbreak Detection

- **The Cat and Mouse Game:** Apps check for Cydia/Frida files.
- **The Bypass:** Use "Shadow" or "Liberty Lite" (Tweaks).
- **Frida Script:**

```
// Hook file existence check
var access = new NativeFunction(Module.findExportByName(null, 'access'), 'int', ['pointer', 'int']);
Interceptor.replace(access, new NativeCallback(function(pathPtr, mode) {
  var path = Memory.readUtf8String(pathPtr);
  if (path.indexOf("Cydia") >= 0) {
    return -1; // File not found
  }
  return access(pathPtr, mode);
}, 'int', ['pointer', 'int']));
```

Android: Bypassing Root Detection

- **SafetyNet / Play Integrity API:** Google's advanced root detection.
- **The Bypass:** MagiskHide / Zygisk.
- **Universal SafetyNet Fix:** A Magisk module that spoofs the device fingerprint to look like a certified, unrooted device.

Deobfuscation

- **ProGuard/R8:** Renames classes to *a.b.c*.
- **The Fix:**
- Look for strings (they usually aren't obfuscated).
- Look for API calls (Android APIs can't be renamed).
- Use JADX "Deobfuscation" mode (renames *a* to *Class001* for clarity).

Patching the APK

- **Scenario:** You can't hook the function, or you want to make the change permanent.
- **Tool:** *apktool*.
- (Disassemble to Smali).
- Edit the *.smali* file (Assembly).
- (Rebuild).
- (Sign with your own key).
- Install.

Smali: The Assembly of Android

- **Registers:** *v0*, *v1*, *p0* (parameter).
- **Opcodes:**
- (If v0 == v1, jump).

Testing GraphQL Endpoints

- **Modern Apps:** Many use GraphQL instead of REST.
- **Introspection:** The feature that lets you ask the API "What queries do you support?"
- **Attack:** If Introspection is enabled, you can dump the entire database schema.
- **Tool:** GraphQL Voyager / Burp Suite GraphQL Raider.

Testing Firebase Security Rules

- **Firebase:** A Backend-as-a-Service.
- **Misconfiguration:** "Allow read/write: if true;"
- **Tool:** *baserunner* or manual checking.
- **Impact:** Anyone can delete the entire database.

Deep Link Exploitation

- **Deep Links:** *app://reset_password?token=123*.
- **Attack:**
  - Create a malicious page.
  - User clicks link.
  - Link triggers the app's exported Activity.
  - App performs sensitive action (e.g., changes password) without checking origin.

WebViews: The Hidden Browser

- **WebView:** A browser inside the app.
- **Risk:** XSS (Cross-Site Scripting).
- **Attack:** If *setJavaScriptEnabled(true)* is on, and the app loads a malicious URL, the attacker can steal cookies or bridge to Java.
- **Bridge:** *addJavascriptInterface* allows JS to call Java functions. RCE risk!

Side Channel: Logcat

- **The Leak:** Developers logging sensitive info for debugging.

Side Channel: Clipboard

- **The Leak:** User copies password from password manager. Malicious app reads clipboard.
- **Android 12+ Fix:** Toast message "App X pasted from your clipboard."
- **Defense:** Mark sensitive fields so they cannot be copied, or clear clipboard on background.

Side Channel: Screenshots

- **The Leak:** OS takes a screenshot when app goes to background (for the "Recent Apps" switcher).
- **Risk:** Sensitive data (credit card, medical info) is saved to disk as an image.
- **Fix:** *FLAG_SECURE*. Prevents screenshots and screen recording.

Final Project: The "Breaker" Report Template

- **Executive Summary:** High level.
- **Scope:** What was tested.
- **Methodology:** Tools used.
- **Findings:**
  - Vulnerability Name.
  - CVSS Score.
  - Proof of Concept (Screenshots/Code).
  - Recommendation.

Final Project: The "Builder" Defense Document

- **Architecture:** How you designed security.
- **Threat Model:** What you are protecting against.
- **Implementation:**
  - Show your *EncryptedSharedPreferences* code.
  - Show your Pinning configuration.
  - Show your ProGuard rules.

Resources for Practice

- **OWASP MSTG Hacking Playground:** A set of vulnerable apps.
- **DVIA (Damn Vulnerable iOS App):** The standard for iOS.
- **InsecureBankv2:** Great for Android.
- **GoatDroid:** Another classic.

Q&A

Questions?