Lecture Plan for Week 10: Hacking Mobile Platforms

Day 1: Introduction to Mobile Platform Security

Slide 1: Title Slide

Hacking Mobile Platforms – Week 10, Day 1

Slide 2: Introduction to Mobile Platform Security

Mobile Platform Security:
Mobile platforms such as Android and iOS have unique security architectures designed to protect user data and prevent unauthorized access.

- Key Security Concerns:
  - Mobile apps often handle sensitive data like personal information, banking details, and location data.
  - Weaknesses in mobile security can lead to data leaks, identity theft, or device compromise.

Slide 3: Legal and Ethical Aspects of Mobile Hacking

- Ethical Hacking:
  Always obtain permission before conducting mobile penetration testing. Unauthorized hacking is illegal and punishable under cybercrime laws.
- Legal Framework:
  - Always comply with laws like GDPR and other data protection regulations when handling user data.
  - Respect privacy and ensure that any hacking activities are done in controlled environments with proper authorization.

Slide 4: Mobile Platform Fundamentals

Android Security Fundamentals:

- Open-source platform.
- Utilizes sandboxing to isolate apps and prevent access to other apps' data.
- Permissions: Apps need explicit permissions to access sensitive data or system resources.

iOS Security Fundamentals:

- Closed-source platform with strict app review processes.
- Utilizes sandboxing, code signing, and App Store approval for apps to prevent malware.
- Data encryption: iOS encrypts data at rest and during communication.

Slide 5: Mobile Application Security Models

- Android Security Model:
  - App Permissions: Users grant permissions during installation or runtime (e.g., location, camera access).
  - Google Play Protect: Scans apps for vulnerabilities and malware.
- iOS Security Model:
  - App Store Review: All iOS apps must pass a strict review process before being allowed on the App Store.
  - Code Signing: Ensures apps are from trusted developers and haven't been tampered with.

Slide 6: Identifying Mobile Security Vulnerabilities

Common Mobile Vulnerabilities:

- Insecure Data Storage: Sensitive data stored without encryption.
- Insecure Communication: Unencrypted traffic between the app and backend server.
- Weak Authentication/Authorization: Lack of proper user authentication and session management.
- Improper Platform Usage: Apps that do not follow platform security guidelines, making them vulnerable.

Slide 7: Tools for Identifying Mobile Vulnerabilities

---

1. Drozer:
   A comprehensive Android security assessment framework.

Example Command:

```
drozer console connect
run app.package.attacksurface <package-name>
```

3. MobSF (Mobile Security Framework):
An open-source mobile app security testing tool for Android and iOS.

4. Frida:
A dynamic instrumentation toolkit for security researchers to trace and manipulate running processes.

Slide 8: Setting Up a Mobile Hacking Environment

To begin mobile security testing, it’s crucial to set up a dedicated environment:

Tools for Android:

1. Android Studio:
   The official IDE for Android development. It provides an emulator for testing Android apps.
2. Genymotion:
   A fast and feature-rich Android emulator ideal for testing apps on different versions of Android.

Tools for iOS:

1. Xcode:
   The official IDE for iOS development. Provides emulators for testing apps on different iPhone models.

Task: Set up Android Studio or Xcode and emulate a mobile device for testing.

Slide 9: Jailbreaking (iOS) and Rooting (Android)

- Jailbreaking:
  The process of removing software restrictions imposed by iOS, allowing users to install unapproved apps, access system files, and modify the system.
- Rooting:
  Gaining superuser access on Android devices, which allows control over the entire device, including system-level files and apps.
- Tools for Jailbreaking:
  - Checkra1n: Popular tool for jailbreaking iOS devices.
- Tools for Rooting Android:
  - Magisk: A tool for rooting Android devices while allowing users to hide root status from apps that check for it.

Slide 10: Device and Emulator Testing

Emulator Testing:

- Emulators simulate mobile environments to test apps without needing a physical device.

Steps:

1. Install Android Studio and create a virtual Android device.
2. Install Xcode and emulate iPhone models for iOS app testing.
3. Test: Perform dynamic analysis and interact with the app's interface to identify vulnerabilities.

Slide 11: Practical Exercise 1: Setting Up a Mobile Hacking Environment

---

Task: Set up an Android Emulator using Android Studio and analyze a vulnerable mobile application.

Instructions:

1. Download and install Android Studio.
2. Create a virtual device (e.g., Pixel 3).
3. Install a vulnerable mobile application (like Damn Vulnerable Mobile App).
4. Test for common vulnerabilities like insecure data storage and improper platform usage.

Slide 12: Practical Exercise 2: Rooting an Android Device

Task: Root an Android emulator and use Drozer to assess its security.

Instructions:

1. Set up an Android emulator using Genymotion.
2. Use Magisk to root the Android emulator.
3. Install Drozer and connect it to the emulator to assess the attack surface of installed apps.

Slide 13: Homework Assignment

---

1. Research Task:
   Write a report on the differences between Jailbreaking iOS and Rooting Android devices, focusing on the security implications of each.
2. Practical Task:
   Install MobSF and analyze an APK file for security vulnerabilities. Document your findings and suggest mitigation strategies.

Slide 14: Conclusion

- Recap:
  Today we covered the fundamentals of mobile platform security, explored tools for identifying vulnerabilities, and set up a mobile hacking environment.

Slide 2: Hour 1 - Analyzing Mobile Apps for Vulnerabilities

Definition:
Analyzing mobile apps for vulnerabilities involves assessing an application to detect security weaknesses in its architecture, code, or configuration.

Common Vulnerabilities:

- Insecure Data Storage: Sensitive information stored in plain text on the device.
- Insecure Communication: Data sent over unencrypted channels.
- Weak Authentication: Insufficient mechanisms for validating users.

Tools:

1. MobSF (Mobile Security Framework):
   A powerful tool for static and dynamic analysis of Android and iOS apps.
2. QARK (Quick Android Review Kit):
   A static analysis tool for Android applications to find common vulnerabilities like insecure permissions.

Task:

- Use MobSF to analyze an APK file for security vulnerabilities.

Slide 3: Hour 2 - Data Storage and Encryption on Mobile Devices

---

Data Storage Security:

- Insecure Data Storage: When apps store sensitive data (passwords, personal information) without encryption, it can easily be accessed if the device is compromised.
- Best Practices:
  - Use Android Keystore or iOS Keychain for sensitive data.
  - Encrypt data using AES encryption.

Example of Insecure Data Storage:

- SharedPreferences in Android storing credentials in plaintext.

Tools for Data Storage Analysis:

- Drozer: Analyze Android applications for insecure storage.

Slide 4: Hour 2 - Insecure Data Transmission

Insecure Data Transmission:

- SSL Pinning: A technique used to ensure an app communicates only with trusted servers by validating SSL certificates.
- Bypassing SSL Pinning:
  - Attackers bypass SSL pinning to intercept sensitive data.

Tools:

1. Burp Suite: Intercepts and manipulates data between the mobile app and the server.
2. Frida: A dynamic instrumentation toolkit used to bypass SSL pinning.

Example of Bypassing SSL Pinning:

```
frida -U -f <app_name> -l bypass_ssl.js --no-pause
```

Slide 5: Hour 3 - Mobile API Testing and Manipulation

Mobile APIs:

- Mobile apps often use APIs to communicate with backend servers. Vulnerabilities in APIs can lead to unauthorized access or data leakage.

Common API Vulnerabilities:

- Lack of Input Validation: APIs that do not validate user input are prone to attacks like SQL Injection.
- Weak Authentication: APIs that do not enforce proper authentication mechanisms.

Tools for API Testing:

1. Postman: A tool to manually test APIs by sending requests to check for vulnerabilities.

2. Burp Suite: Can be used to automate API fuzzing and detect vulnerabilities.

Example:
Testing an API endpoint with invalid input:

```
POST /api/login HTTP/1.1
Host: target.com
Content-Type: application/json
{"username": "' OR '1'='1", "password": ""}
```

Slide 6: Hour 3 - Reverse Engineering Mobile Apps

Reverse Engineering:
The process of decompiling an APK or IPA file to analyze its source code, structure, and internal components.

- Purpose: To identify security flaws, hidden functionality, or hardcoded sensitive information.

Tools:

1. JADX: A tool for decompiling Android APK files into readable Java code.
    - Command:

```
jadx <apk_file>
```

Ghidra: An open-source reverse engineering tool that supports multiple platforms including mobile applications.

Slide 7: Hour 4 - Exploiting Authentication and Authorization Flaws

---

Authentication and Authorization Flaws:
Mobile apps often contain weaknesses in their authentication mechanisms that allow attackers to bypass login or access privileged information.

- Common Flaws:
    - Weak Password Policies.
    - Hardcoded API Keys.
    - Lack of Token Expiration.

Example:

- Weak Authentication: An attacker may bypass login using poorly implemented session tokens.

Tools:

1. Burp Suite: For manipulating HTTP requests and testing authentication mechanisms.
2. Frida: For intercepting function calls within the app and testing its security.

Slide 8: Hour 4 - In-App Purchases and License Verification Bypass

---

In-App Purchases (IAP) and License Verification:
Attackers exploit flaws in app payment systems to unlock paid features without proper authorization.

Techniques:

1. Tampering with Local Data: Modifying app data to simulate a successful purchase.
2. Intercepting Communication: Modifying requests between the app and the server to bypass verification.

Tools:

1. Lucky Patcher: A tool that modifies app permissions and bypasses license verification.
2. Frida: Can be used to hook into functions that manage in-app purchases and manipulate the verification process.

Task:

- Use Frida to intercept and modify an in-app purchase request.

Slide 9: Practical Exercise 1: Analyzing Mobile App Security with MobSF

Task:
Analyze a mobile app APK with MobSF to identify common vulnerabilities (insecure storage, weak authentication, etc.).

Instructions:

1. Download MobSF and the APK file.
2. Upload the APK to MobSF and run a static analysis.
3. Document vulnerabilities like hardcoded keys or insecure storage methods.

Slide 10: Practical Exercise 2: Reverse Engineering an APK

Task:
Reverse engineer an Android APK using JADX to analyze the source code and identify any hardcoded sensitive information.

Instructions:

1. Decompile an APK file with JADX.
2. Analyze the decompiled Java code for hardcoded API keys or credentials.
3. Document any security issues found.

Slide 11: Homework Assignment

---

1. Research Task:
   Write a report on the process of SSL Pinning Bypass and the security risks associated with Insecure Data Transmission in mobile applications.
2. Practical Task:
   Perform API testing on a vulnerable mobile app and document any issues with authentication or input validation using Postman and Burp Suite.

Slide 12: Conclusion

- Recap:
  Today we covered the analysis of mobile app vulnerabilities, reverse engineering, API testing, and exploiting authentication and in-app purchase flaws.
- Next Topic:
  In the next session, we will continue with advanced mobile app exploitation techniques and defensive strategies for securing mobile applications.

Slide 2: Hour 1 - Tampering with Mobile App Logic

Definition:
Tampering with mobile app logic refers to modifying the internal functioning of an application to bypass restrictions, unlock features, or exploit vulnerabilities.

Common Scenarios:

- Bypassing authentication mechanisms.
- Unlocking paid features or in-app purchases.
- Manipulating app behavior for personal gain.

Techniques:

- Code Injection: Modifying the app's code at runtime.
- Binary Patching: Changing the compiled code of the app to alter its behavior.

Tools for Tampering:

- Frida: A dynamic instrumentation toolkit used to intercept and manipulate app behavior at runtime.
- APKTool: For decompiling, modifying, and recompiling Android apps.

Slide 3: Hour 2 - Mobile Malware and Spyware

Mobile Malware:
Malware designed to target mobile devices to steal data, spy on users, or damage the device. It can come in the form of apps, phishing links, or malicious files.

Types of Mobile Malware:

1. Trojan Horses: Apps that appear legitimate but carry malicious code.
2. Ransomware: Locks users out of their devices and demands a ransom.
3. Spyware: Tracks user activity without their knowledge, such as call logs, text messages, and location data.

Real-World Examples:

- Pegasus Spyware: A sophisticated spyware used to target iPhones and Android devices, capable of extracting vast amounts of data from a device without the user knowing.
- Joker Malware: Found in apps on Google Play, Joker malware steals SMS messages, contact lists, and device information.

Techniques for Deploying Malware:

- Malicious Apps: Disguising malware as a legitimate app and distributing it via third-party stores.
- Phishing: Sending malicious links via SMS, email, or social media that download malware onto the device.

Slide 4: Hour 2 - Real-World Mobile Exploits and Vulnerabilities

---

Real-World Mobile Exploits:

- Stagefright Exploit: A vulnerability in the Android media processing library that allowed remote code execution via MMS messages.
- BlueBorne Vulnerability: A Bluetooth-based vulnerability that allowed attackers to gain control of devices without user interaction.

Common Vulnerabilities:

1. Insecure Data Storage: Sensitive information stored in plaintext or unprotected locations.
2. Insecure Communication: Lack of encryption in data transmission between the app and the server.
3. Weak Authentication and Session Management: Applications that fail to enforce strong user authentication or fail to expire sessions properly.

Tools for Exploiting Vulnerabilities:

- Metasploit Framework: Used for discovering, testing, and exploiting vulnerabilities in mobile devices and applications.
- Drozer: A security testing framework for Android applications, ideal for finding vulnerabilities in app components.

Slide 5: Hour 3 - Detecting and Preventing Mobile Exploits

Detecting Mobile Exploits:

- Mobile Threat Detection: Use of threat detection apps that scan devices for malware, unauthorized apps, or unusual activity.
- Network Traffic Analysis: Monitoring the communication between mobile apps and servers to detect anomalies or unencrypted data.

Tools for Detection:

- Wireshark: Used for analyzing mobile traffic and identifying unencrypted communication.
- Mobile Device Management (MDM): Solutions that enforce security policies on corporate mobile devices.

Preventing Mobile Exploits:

1. Regular Software Updates: Ensures that vulnerabilities are patched as soon as possible.
2. Use of Mobile Security Solutions: Deploying apps and systems that continuously monitor for threats.
3. Application Hardening: Implementing security mechanisms within the app to make it more resilient to tampering and exploits.

Slide 6: Hour 4 - Mobile Application Security Best Practices

Best Practices for Mobile App Security:

1. Data Encryption: Encrypt sensitive data both at rest and in transit using strong encryption algorithms.

2. Secure Authentication: Implement multifactor authentication (MFA) and secure password management.
3. Secure Code Development: Follow secure coding standards to minimize vulnerabilities like buffer overflows and SQL injection.
4. Regular Security Audits: Conduct periodic code reviews and vulnerability assessments to identify and fix potential issues.
5. Implementing SSL/TLS: Ensure that all communication between the app and server is encrypted to prevent man-in-the-middle (MITM) attacks.

Tools for Implementing Best Practices:

- OWASP Mobile Security Project: A resource that provides guidelines and tools to help secure mobile applications.
- ProGuard: A tool used to obfuscate and shrink Android app code, making reverse engineering more difficult.

Slide 7: Hour 4 - Ethical Use of Mobile Hacking Skills

Ethical Considerations:

- Responsible Disclosure: When identifying a vulnerability, always inform the app developers or responsible party without exploiting it for malicious purposes.
- Legal Compliance: Ensure that all penetration testing and security assessments are done within the legal boundaries and with proper authorization.
- User Privacy: Never access or misuse personal data during mobile hacking exercises or assessments.

Real-World Example:

- Bug Bounty Programs: Ethical hackers are rewarded for responsibly disclosing vulnerabilities in mobile applications and platforms through programs like HackerOne and Bugcrowd.

Slide 8: Practical Exercise 1: Tampering with Mobile App Logic

---

Task:
Tamper with the logic of a vulnerable mobile app using Frida to bypass authentication.

Instructions:

1. Set up a mobile app in an emulator.
2. Use Frida to intercept and modify function calls that handle authentication.
3. Test to see if the app's authentication can be bypassed.

Slide 9: Practical Exercise 2: Detecting Mobile Malware

Task:
Analyze a sample Android APK for signs of malware using a combination of MobSF and Wireshark.

Instructions:

1. Upload the APK to MobSF for static analysis.
2. Monitor network traffic using Wireshark while the app is running in an emulator.
3. Identify any suspicious behavior or unencrypted communication.

Slide 10: Homework Assignment

---

1. Research Task:
   Write a report on two real-world mobile malware cases and explain how they exploited mobile vulnerabilities.
2. Practical Task:
   Set up Metasploit and use it to test a mobile device (or emulator) for vulnerabilities related to insecure communication and authentication flaws.

Slide 11: Conclusion

- Recap:
  Today we covered mobile app tampering, malware analysis, real-world mobile exploits, and best practices for mobile security.
- Next Topic:
  In the next session, we will delve deeper into mobile device forensics and advanced mobile security techniques.

Slide 2: Hour 1 - Mobile Hacking Challenges and CTFs

Mobile Hacking Challenges:
Capture the Flag (CTF) events and hacking challenges provide environments for testing mobile hacking skills in a controlled setting. These platforms offer practical, hands-on opportunities for participants to exploit vulnerabilities in real-world mobile systems.

- Popular CTF Platforms:
  - HackTheBox Mobile Lab
  - Root-Me
  - Mobile CTFs (e.g., PicoCTF, OWASP Mobile CTF)

Challenges Include:

- Finding and exploiting mobile vulnerabilities.
- Reverse engineering APK or IPA files.
- Bypassing mobile app security mechanisms.

Slide 3: Hour 1 - iOS Jailbreaks and Bypassing Security Features

---

iOS Jailbreaking:
Jailbreaking is the process of removing software restrictions imposed by Apple, allowing users to install unauthorized apps, tweak system features, and bypass security mechanisms.

Types of Jailbreaking:

1. Untethered Jailbreak: The device remains jailbroken even after rebooting.
2. Tethered Jailbreak: Requires connecting to a computer after each reboot to maintain the jailbreak.

Jailbreaking Tools:

- Checkra1n
- Unc0ver

Bypassing Security Features:

- Gaining root access to the device.
- Accessing system files and sensitive data.
- Installing apps not authorized by the App Store.

Real-World Example:
Pegasus spyware took advantage of jailbreaking vulnerabilities to access iOS devices and steal sensitive data.

Slide 4: Hour 2 - Android Rooting and Custom ROMs

Android Rooting:
Rooting an Android device allows users to gain root (superuser) access, providing full control over the operating system. This enables users to bypass restrictions imposed by the manufacturer, install custom software, and modify system files.

Custom ROMs:
A custom ROM replaces the stock Android operating system on a device with a version created by developers, offering additional features or improvements over the default OS.

Benefits of Rooting:

- Full control over the device.
- Ability to modify or remove system applications.
- Installing apps that require root access (e.g., firewalls, custom tweaks).

Risks of Rooting:

- Increased vulnerability to malware.
- Void of manufacturer warranty.
- Potential for system instability or bricking the device.

Tools for Rooting:

- Magisk
- SuperSU

Slide 5: Hour 2 - Mobile App Debugging and Patching

Mobile App Debugging:
Debugging is the process of testing and troubleshooting the mobile app's code to identify and resolve vulnerabilities. Hackers can leverage debugging to identify weaknesses in the app's logic or security implementations.

Debugging Techniques:

- Monitoring the app's behavior in real-time.
- Tracking memory usage, network calls, and file system access.

Patching Mobile Apps:
Patching involves modifying the app's binary code or configuration to bypass security features, unlock premium features, or manipulate app behavior.

Tools for Debugging and Patching:

- Frida
- Xposed Framework

Common Use Cases:

- Bypassing in-app purchases or ads.
- Unlocking premium content in games or apps.
- Analyzing how the app interacts with the operating system and backend servers.

Slide 6: Hour 3 - Application Security Testing on Real Devices

---

Security Testing on Real Devices:
While emulators provide a controlled environment, testing on real mobile devices is crucial for identifying issues that may not be replicated in virtual environments, such as hardware-specific vulnerabilities or OS version differences.

Benefits of Testing on Real Devices:

- Identifies hardware-related vulnerabilities.
- Tests real-world performance under typical usage conditions.
- Exposes potential issues with different Android/iOS versions.

Tools for Application Security Testing:

- OWASP Mobile Security Testing Guide (MSTG) framework.
- ZAP (Zed Attack Proxy) for testing API calls and backend interactions.

Test Scenarios:

- Testing network traffic for data leakage or unencrypted communication.
- Verifying permissions and access control mechanisms.
- Evaluating the app's resistance to reverse engineering.

Slide 7: Hour 3 - Malware Analysis on Mobile Platforms

Malware Analysis on Mobile Devices:
Mobile malware analysis is the process of understanding the behavior, code, and impact of malicious apps targeting mobile platforms. This includes investigating how malware spreads, its impact on the system, and potential mitigation techniques.

Types of Mobile Malware:

- Spyware: Steals personal data, tracks user activity.
- Ransomware: Locks device functionality and demands payment.
- Trojans: Disguised as legitimate apps but perform malicious actions in the background.

Steps in Mobile Malware Analysis:

1. Static Analysis: Examining the malware's code without executing it.
2. Dynamic Analysis: Running the malware in a sandbox environment to observe its behavior.

Tools for Mobile Malware Analysis:

- MobSF (Mobile Security Framework) for static and dynamic analysis.
- Apktool for reverse engineering Android apps.

Real-World Malware:

- Joker malware found on the Google Play Store, which stole SMS and contact data from infected devices.

Slide 8: Hour 4 - Mobile Device Forensics

Mobile Device Forensics:
Mobile forensics involves extracting, analyzing, and preserving digital evidence from mobile devices to support legal investigations.

Types of Data Extracted:

- Call logs, SMS, and MMS.
- Photos, videos, and audio files.
- Location data, GPS coordinates.
- Social media and app usage.

Forensics Techniques:

- Logical Acquisition: Extracting data from the file system without accessing the physical memory.
- Physical Acquisition: Extracting data directly from the device's memory (e.g., flash memory).

Forensic Tools:

- Cellebrite: A mobile forensics tool for data extraction and analysis.
- UFED (Universal Forensic Extraction Device) for iOS and Android devices.

Challenges in Mobile Forensics:

- Encryption: Modern devices use strong encryption, making it difficult to extract meaningful data.
- Data Overwrite: Deleted data can be overwritten by new data, making it unrecoverable.

Best Practices for Mobile Forensics:

- Isolate the device from external networks to prevent remote wiping.
- Ensure data integrity by creating hashes of extracted data.
- Use tools that support chain-of-custody documentation for legal proceedings.

Slide 9: Practical Exercise 1: Jailbreaking and Rooting

Task:
Root an Android device and jailbreak an iOS device to gain administrative control and analyze the security features of each operating system.

Instructions:

1. Use Magisk to root an Android device.
2. Use Checkra1n to jailbreak an iOS device.
3. Document the security features that can be bypassed after rooting/jailbreaking.

Slide 10: Practical Exercise 2: Application Security Testing on Real Devices

---

Task:
Perform a security assessment on a real mobile device by testing a vulnerable application for common issues like insecure data storage and unencrypted network traffic.

Instructions:

1. Install a vulnerable mobile application (e.g., DVIA or Damn Vulnerable Mobile App) on a real device.
2. Analyze the app’s data storage, permissions, and network traffic for vulnerabilities.
3. Document any issues found and suggest mitigation techniques.

Slide 11: Homework Assignment

---

1. Research Task:
   Write a report comparing Android rooting and iOS jailbreaking, focusing on the security implications, legal aspects, and user risks.
2. Practical Task:
   Conduct a malware analysis on a sample APK file using MobSF. Document any suspicious behavior or permissions used by the app and explain the potential impact on user privacy.

Slide 12: Conclusion

- Recap:
  Today we covered advanced mobile hacking techniques including jailbreaking, rooting, mobile app debugging, malware analysis, and mobile forensics.
- Next Topic:
  In the next session, we will dive deeper into mobile network attacks and advanced exploitation techniques.