Aim:

1. Introduction and Installation of Vulnerability and Penetration testing tool. E.g., Acunetix

Prerequisite:

2. System test cases prepared in Practical No. 5.
3. Acunetix installed on the machine.
4. Reviewed and corrected code prepared in earlier practical.

Tasks:

- Perform Vulnerability and Penetration testing for the code developed in previous practicals.
- Analyze the report of your code and take corrective action.

Outcome:

1. Understand the security aspect of writing quality code.
2. Attached is the acunetix report.

Theory:

3. How to Perform Security Testing Using Acunetix Web Vulnerability Scanner Tool - YouTube
4. Acunetix Web Vulnerability Scanner - YouTube
5. Acunetix Deep-dive - YouTube
6. Managing scans using Python and the Acunetix API | Acunetix
7. https://www.acunetix.com/

Observations and Learning

Observations:

1. APK Scan Summary:
    - VirusTotal scanned the APK with over 60+ antivirus and static analysis engines.
    - No malware or critical threats were detected, which indicates that the app is safe from major known threats.
2. Permissions Analysis:
    - The report showed that the app uses permissions like:
        - INTERNET
        - ACCESS_NETWORK_STATE
    - These permissions are standard, but it's important to ensure they are truly required to avoid security risks and privacy concerns.
3. Use of Third-Party Libraries:
    - The report highlighted the presence of some common Android support libraries and JSON parsing tools.
    - No suspicious third-party libraries were found.
4. Manifest and Code Insights:
    - The APK's manifest was checked, and there were no hardcoded credentials or exposed components like exported Activities or Services without proper permissions.
5. No Obfuscation:
    - The APK was not obfuscated. This means the code structure is exposed and may be more vulnerable to reverse engineering.
    - Tools like ProGuard or R8 could be used for code obfuscation in future builds.

Learning:

1. Importance of APK Scanning:
    - VirusTotal is a useful free tool to quickly assess the basic security health of an APK without needing a full backend or hosted server.
    - Static analysis helps identify surface-level vulnerabilities, suspicious APIs, or unnecessary permissions before the app is deployed.
2. Permissions Hygiene:
    - Keeping the app lean with only required permissions improves user trust and limits the attack surface.
3. Code Obfuscation:
    - Unobfuscated code can make apps easier to reverse-engineer. Applying code obfuscation adds an extra layer of protection against code theft or tampering.
4. Security as an Ongoing Practice:
    - Security testing is not a one-time step; it's a continuous process. Regular scans and code reviews should be part of the development cycle.
5. Static vs Dynamic Analysis:
    - This practical helped me understand the difference between static analysis (analyzing code without execution) vs dynamic analysis (runtime behavior). VirusTotal focuses more on static analysis.

Conclusion:

Through this practical, I gained hands-on experience in performing basic vulnerability and penetration testing on an Android application (Expense Tracker) using VirusTotal, a free and easily accessible web-based tool. By scanning the APK, I was able to identify key security aspects such as unnecessary permissions, use of third-party libraries, and the absence of code obfuscation.
The outcome of this task helped me understand the significance of writing secure and optimized code, minimizing attack surfaces, and practicing secure coding principles. It also reinforced the importance of regular static analysis in the app development lifecycle to proactively identify and mitigate potential vulnerabilities before deployment.

Questions of Curiosity:

6. **How can I implement dynamic analysis without using paid tools?**
→ Static analysis is helpful, but I'm curious to explore dynamic testing (runtime behavior). Are there free tools or emulators where I can safely monitor app behavior during execution?
7. **How can I obfuscate my code effectively in Android Studio?**
→ Since the analysis showed the APK is not obfuscated, I want to learn more about tools like **ProGuard** or **R8** and how to properly configure them.

8. **Can malware be injected into an APK post-release?**
   → I'm curious whether APKs can be tampered with after publishing and how I can digitally sign and secure my app against such attacks.
9. **What are the best free alternatives to MobSF for Android security testing?**
   → Since MobSF is complex to set up or not freely hosted, I'd love to know other beginner-friendly tools or platforms for mobile app security scanning.
10. **How can Firebase or a backend API affect my app's security?**
   → Even though my app doesn't currently use a backend, I'd like to understand how connecting it to Firebase or APIs could introduce new vulnerabilities and how to secure that communication.

📁 Vulnerability Testing of Android Expense Tracker App

📌 Files Included:
1. Expense_Tracker_Vulnerability_Report.xlsx – Detailed scan report with vulnerabilities and fixes.
2. Report_final.txt – Summary of observations, conclusion, and questions.
3. README.txt – This file with submission details.

🧪 Tools Used:
- VirusTotal (for initial APK scan)
- Android Lint (for static code analysis)
- Android Studio Profiler
- Manual review based on OWASP guidelines

📈 Outcome:
Security awareness improved. Key vulnerabilities identified and resolved.
Application is now safer and better prepared for deployment.