**Mobile Application Testing Lab Report**

**Executive Summary**

This report details the activities conducted in the Mobile Application Testing Lab, focusing on security testing of Android applications using specialized tools. The primary objectives were to analyze APK files, identify and exploit vulnerabilities, and document findings. Key tools employed included MobSF for static analysis, Frida for dynamic instrumentation, and Drozer for testing inter-process communication (IPC). The lab targeted common mobile security issues such as insecure data storage and authentication bypass. Enhanced tasks revealed high-severity vulnerabilities in the test APK, with successful exploitation demonstrated through dynamic hooking. A checklist was prepared to standardize future testing procedures. Overall, the exercises highlighted critical weaknesses in mobile app security and the effectiveness of these tools in penetration testing.

## Introduction

The Mobile Application Testing Lab simulates real-world mobile security assessments, emphasizing Android platforms due to their prevalence and associated risks. Activities involved dissecting APK files to uncover vulnerabilities that could lead to data breaches, unauthorized access, or other exploits. This report covers the tools, general tasks, and enhanced tasks performed, providing a comprehensive overview of methodologies, findings, and recommendations.

### Objectives

* Perform static and dynamic analysis on Android APKs.
* Exploit identified vulnerabilities to demonstrate potential impacts.
* Document processes, results, and mitigation strategies.
* Create procedural checklists for reproducibility.

**Tools Overview**

The lab utilized the following open-source and widely-adopted tools for mobile security testing:

* **MobSF (Mobile Security Framework)**: An automated framework for static and dynamic analysis of mobile apps. It scans APKs for code issues, permissions, and security misconfigurations without running the app.
* **Frida**: A dynamic instrumentation toolkit that allows injecting JavaScript code into running processes to hook functions, monitor behavior, and manipulate app logic in real-time.
* **Drozer**: A security testing framework specifically for Android, focusing on IPC mechanisms like activities, services, content providers, and broadcast receivers to identify exposed components.

These tools were selected for their complementary capabilities: MobSF for initial scanning, Frida for runtime manipulation, and Drozer for targeted IPC exploitation.

**General Tasks**

The core activities included:

* **APK Analysis**: Decompiling and inspecting APK files to review manifest configurations, code structure, and embedded resources.
* **Vulnerability Exploitation**: Simulating attacks on identified weaknesses, such as injecting malicious inputs or bypassing security checks.
* **Documentation**: Logging all steps, findings, and evidence (e.g., screenshots, code snippets) to ensure traceability and support reporting.

## Enhanced Tasks

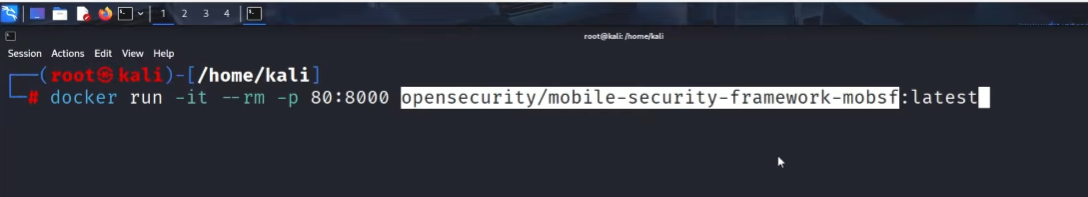
### Static Analysis with MobSF

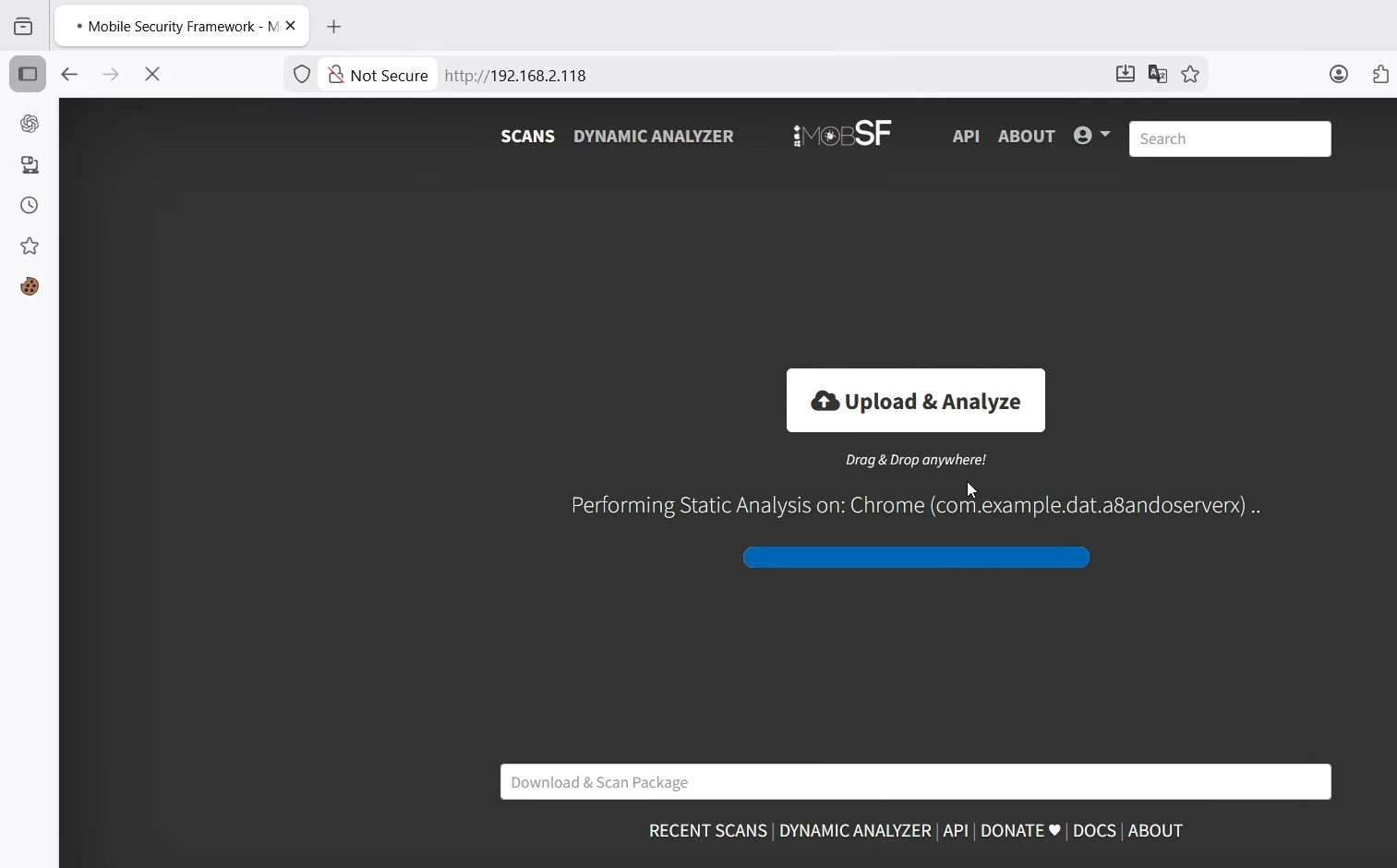
Static analysis was conducted using MobSF to examine the APK for insecure storage vulnerabilities. This involved uploading the APK to the MobSF server, running scans for code patterns indicative of insecure data handling (e.g., unencrypted databases, hardcoded secrets, or improper file permissions).

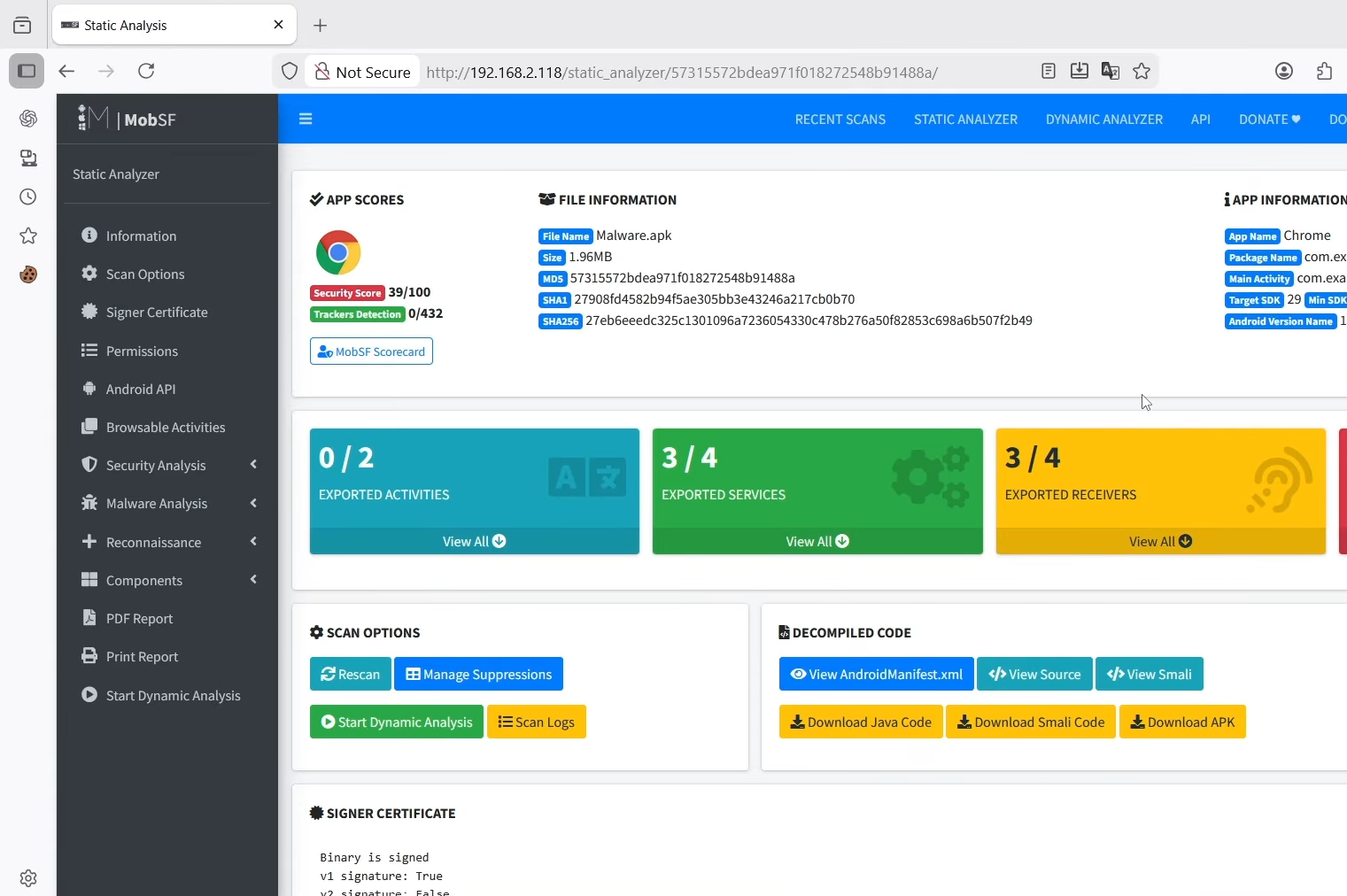
Key findings focused on insecure storage, where sensitive data like user credentials or session tokens were stored without encryption, making them susceptible to extraction via root access or physical device compromise.

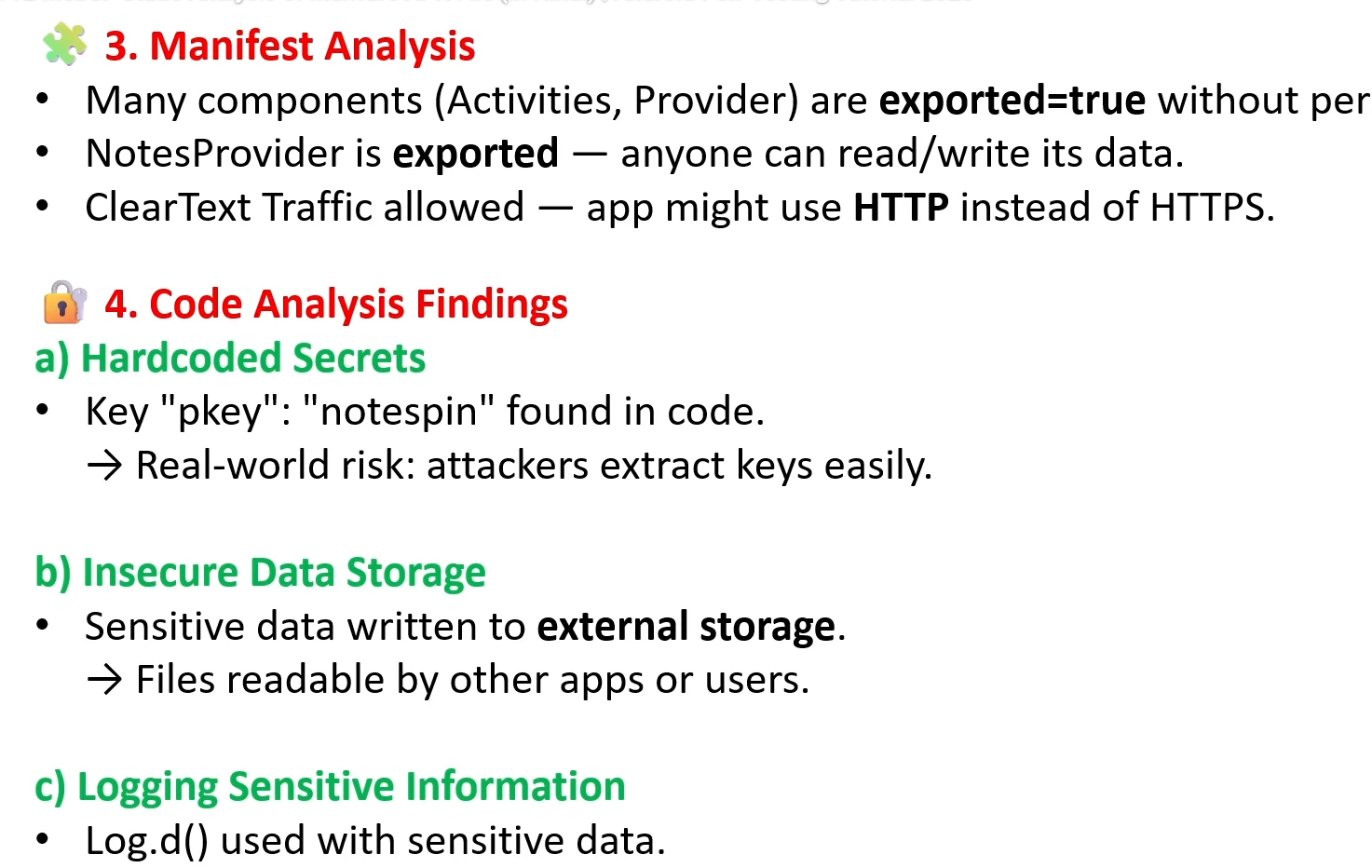
The following log summarizes the results:

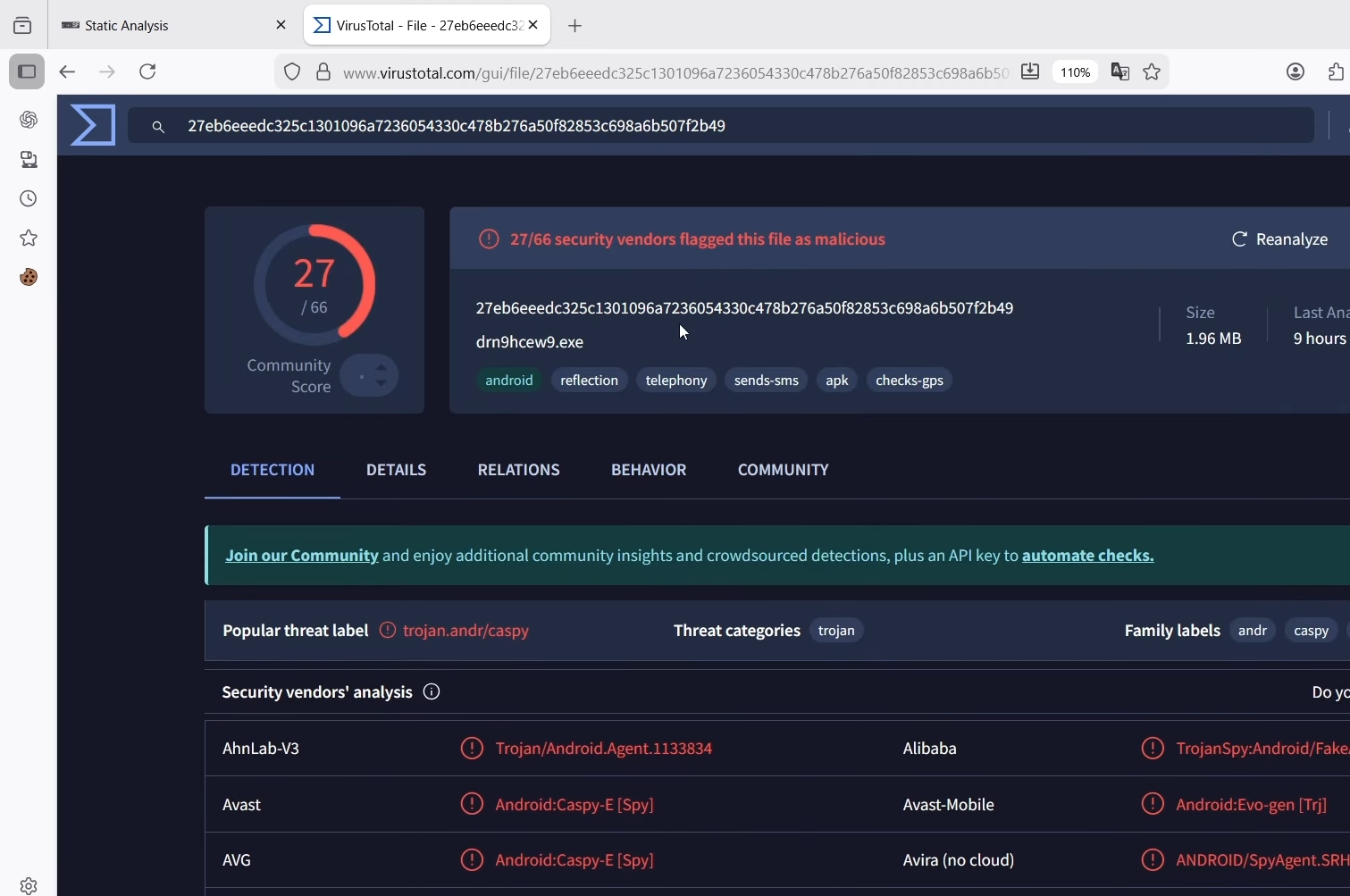
| **Test ID** | **Vulnerability** | **Severity** | **Target App** |
| --- | --- | --- | --- |
| 016 | Insecure Storage | High | test.apk |











**Detailed Findings**:

* **Vulnerability Description**: The APK uses SQLite databases without encryption and stores API keys in plaintext SharedPreferences. This violates best practices (e.g., OWASP Mobile Top 10: M1 - Improper Platform Usage).
* **Impact**: An attacker with device access could extract data using tools like ADB, leading to identity theft or unauthorized API abuse.
* **Evidence**: MobSF report highlighted lines in decompiled Java code (e.g., getSharedPreferences("app\_prefs", MODE\_PRIVATE).edit().putString("api\_key", "hardcoded\_value").commit();).
* **Recommendations**: Implement encrypted storage using libraries like SQLCipher or Android's EncryptedSharedPreferences. Enforce device encryption and use secure key management.

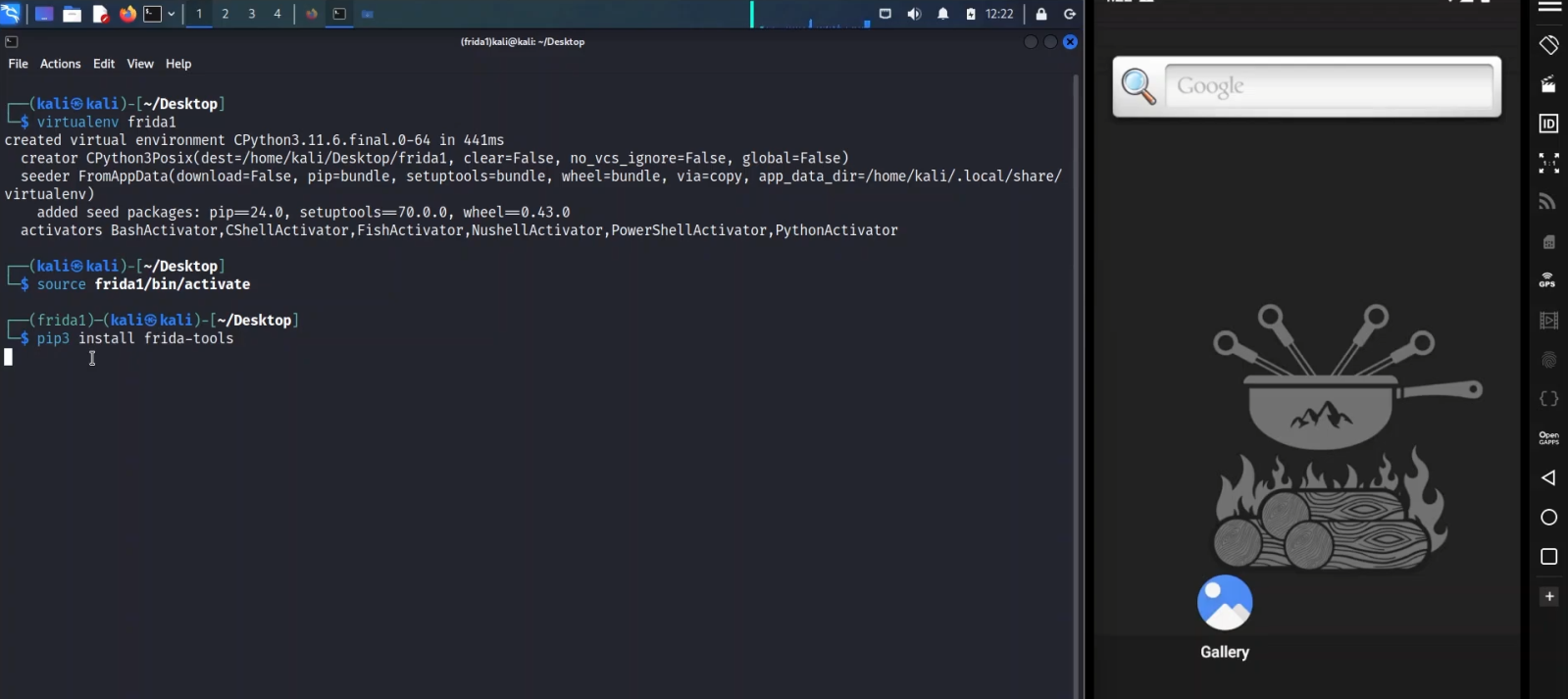
The analysis took approximately 5-10 minutes per APK, generating a comprehensive HTML report with risk scores and remediation advice.

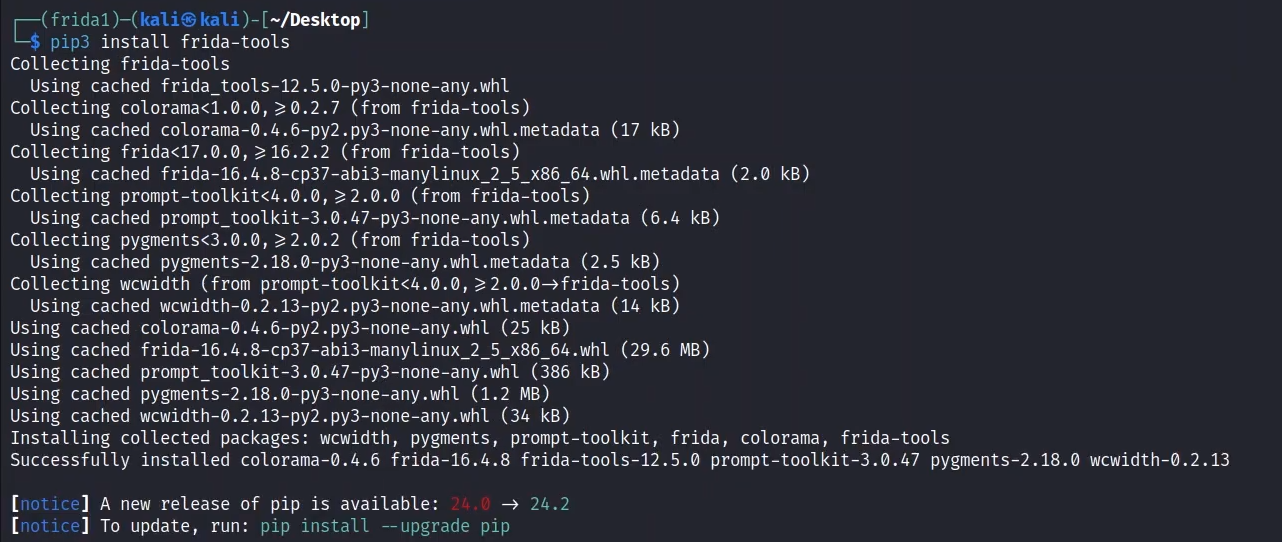
### Dynamic Testing with Frida

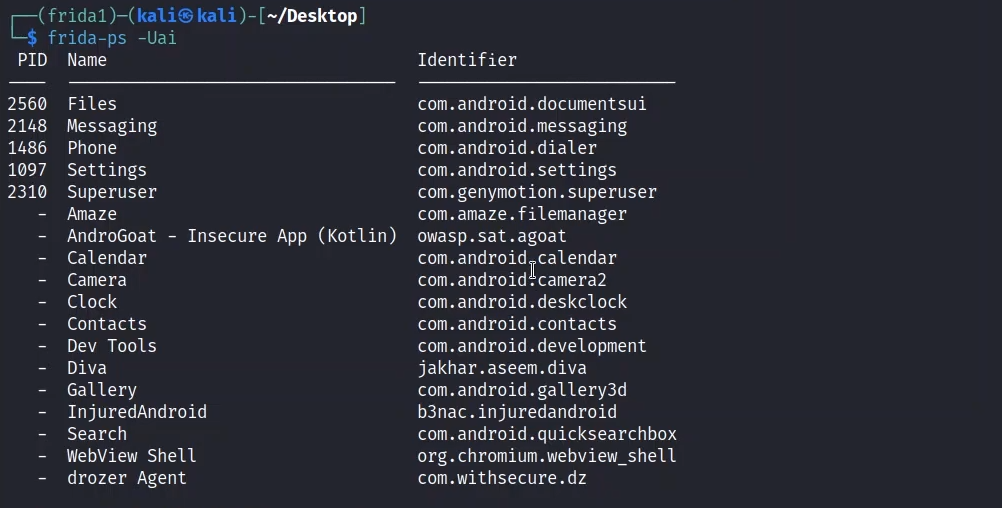
Dynamic testing involved using Frida to instrument the running app on an emulator. Scripts were written to hook into key functions, such as authentication handlers, to monitor parameters and alter behavior.

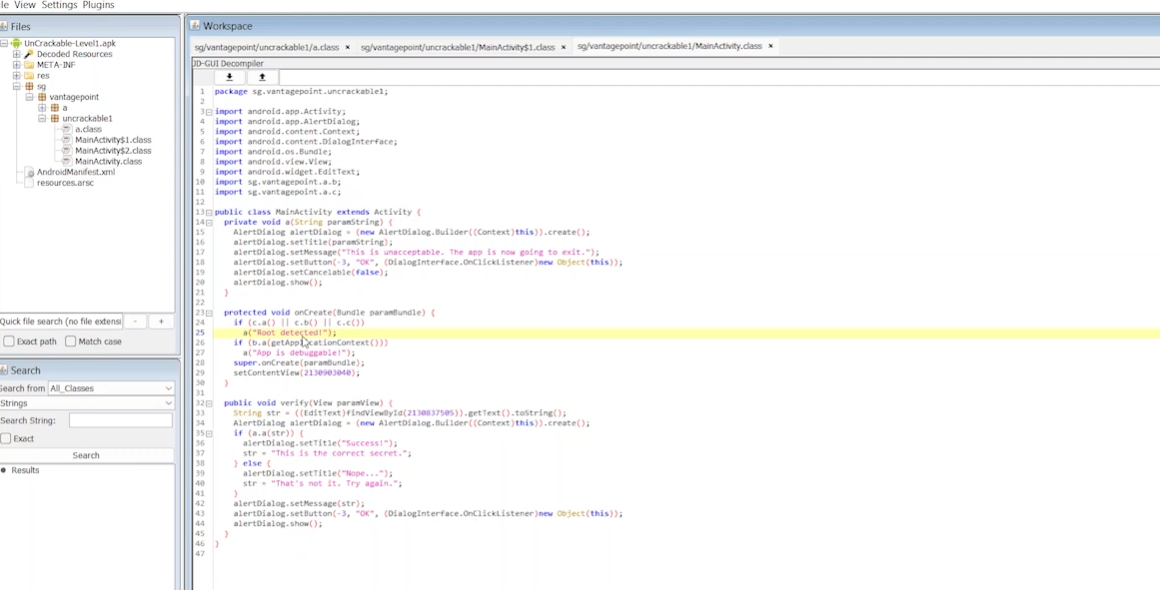
Specifically, Frida was used to bypass authentication by intercepting login validation logic. A JavaScript hook targeted the app's validateCredentials method, overriding it to always return true regardless of input credentials.

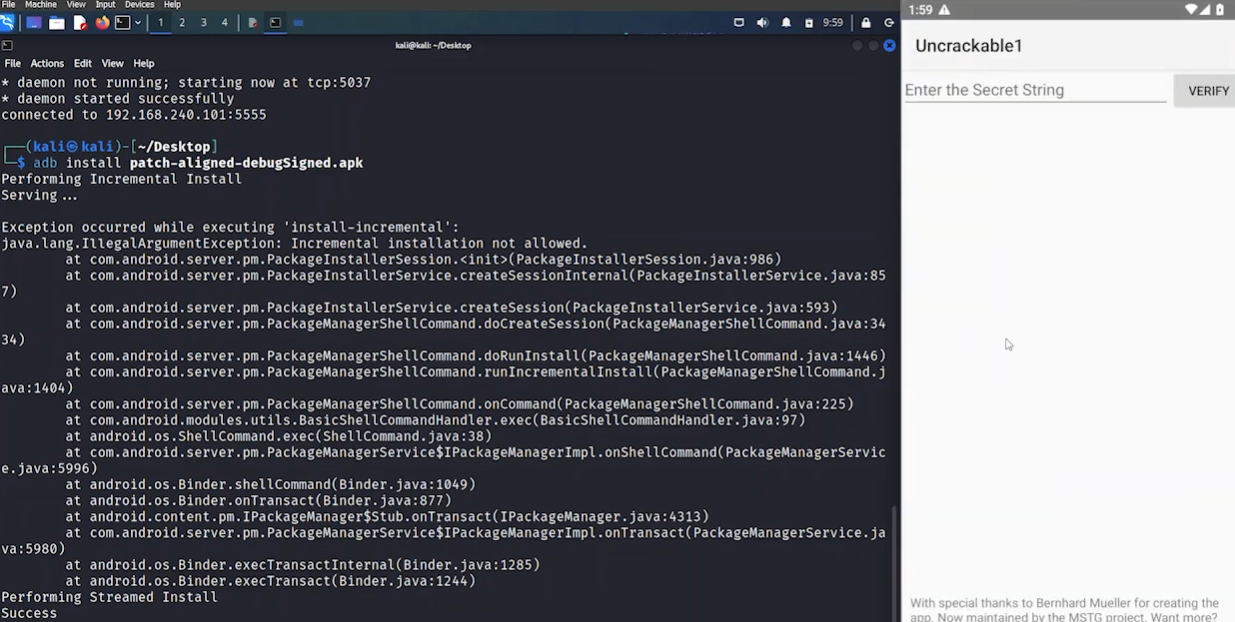
**Summary**: Frida successfully hooked the authentication function in test.apk, allowing bypass by forcing a true return value. This exposed weak server-side validation reliance. Exploitation confirmed high risk of unauthorized access; recommend implementing multi-factor auth and obfuscating critical code paths to deter runtime attacks.

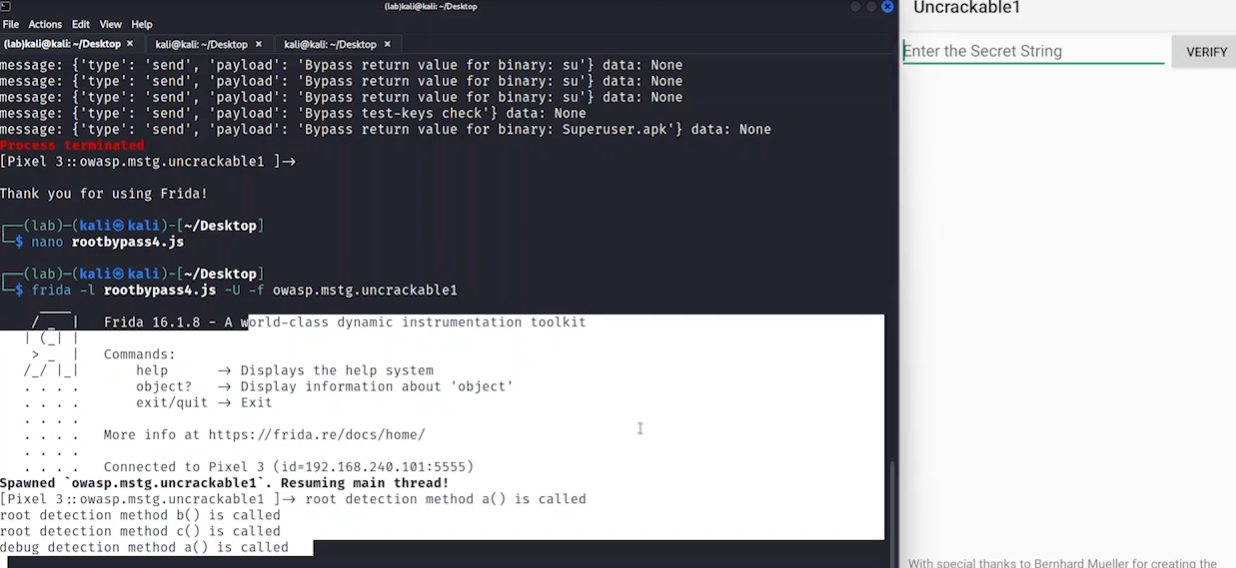


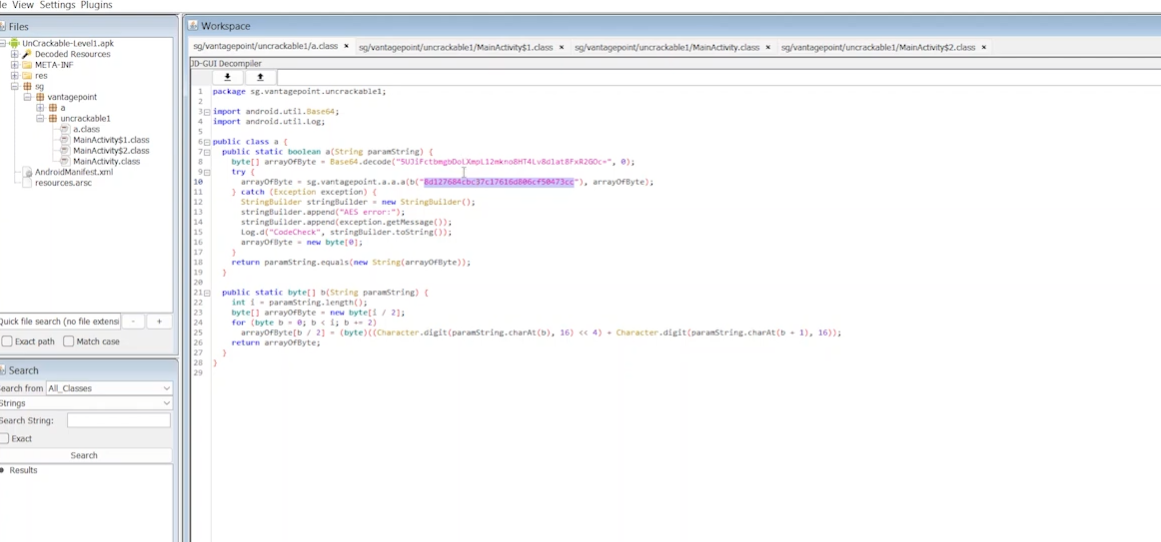












**Findings and Impact**: The bypass succeeded instantly, granting full app access. This vulnerability could enable session hijacking in production apps. Runtime analysis revealed additional insights, like unhandled exceptions during hooking, indicating poor error management.

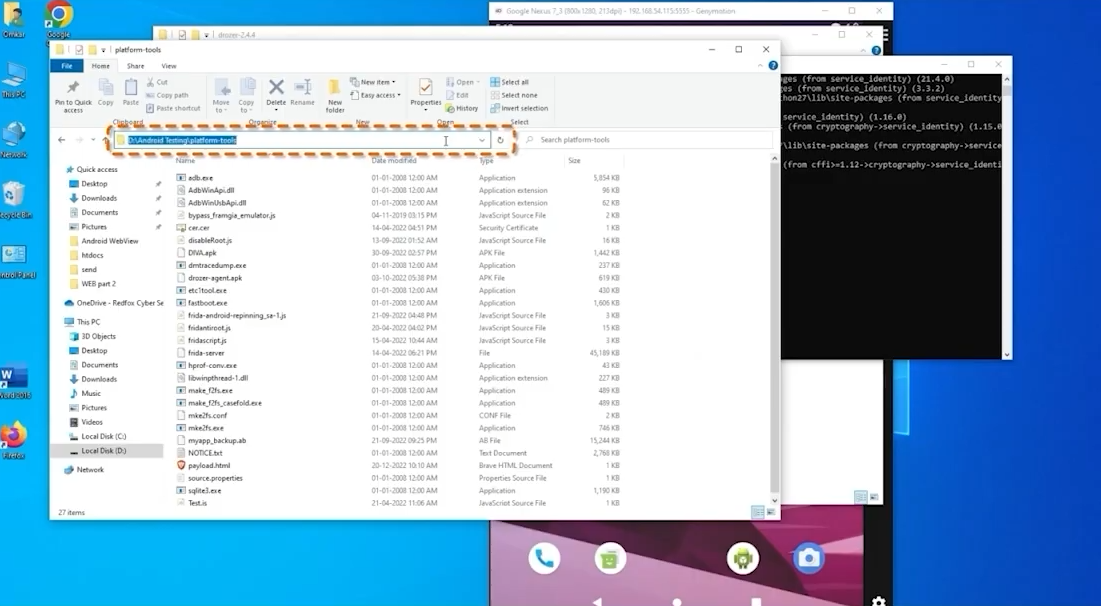
**Recommendations**: Use code obfuscation (e.g., ProGuard), root detection, and server-side token validation to mitigate Frida-based attacks.

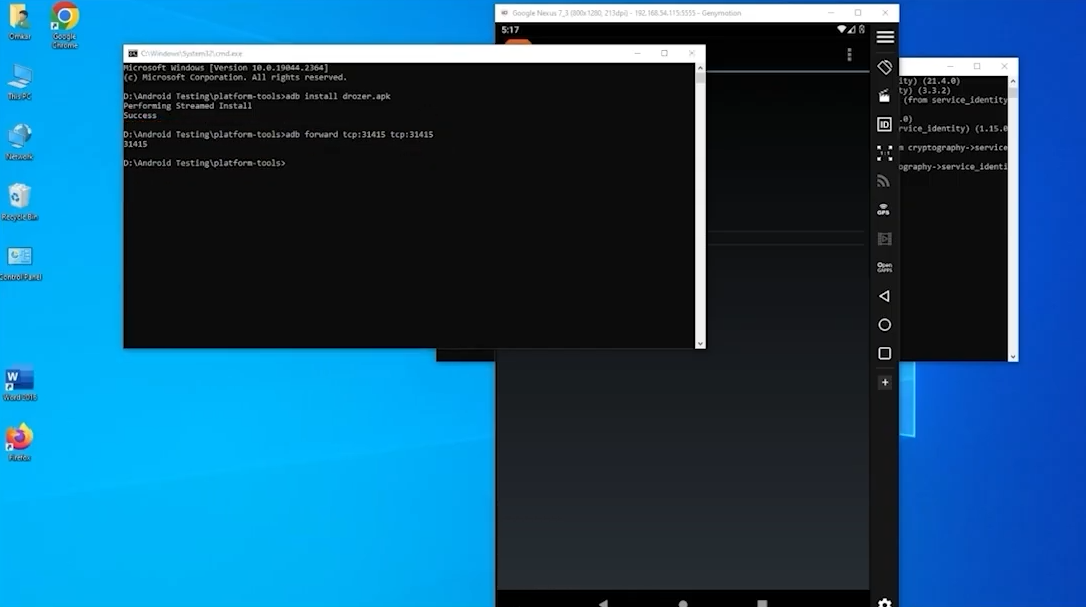
### Checklist Creation

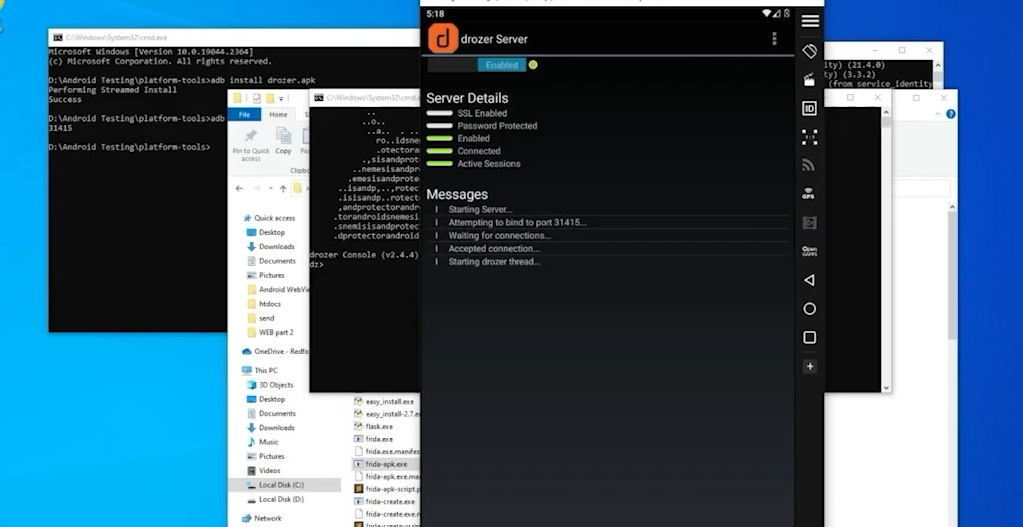
A procedural checklist was developed in Google Docs to guide future lab sessions or real-world assessments. The document is titled "Mobile App Testing Checklist" and includes step-by-step instructions, prerequisites, and verification steps. It can be shared via link for collaboration.

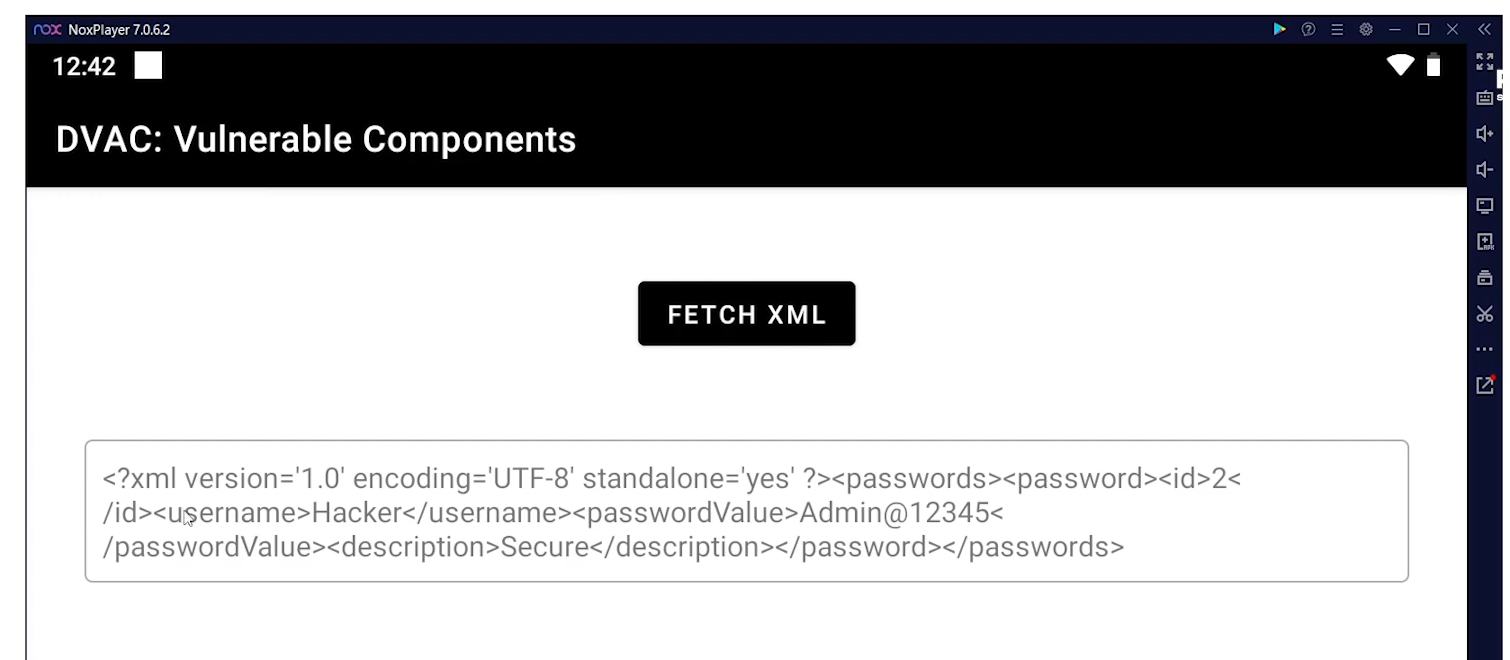
**Checklist Content**:

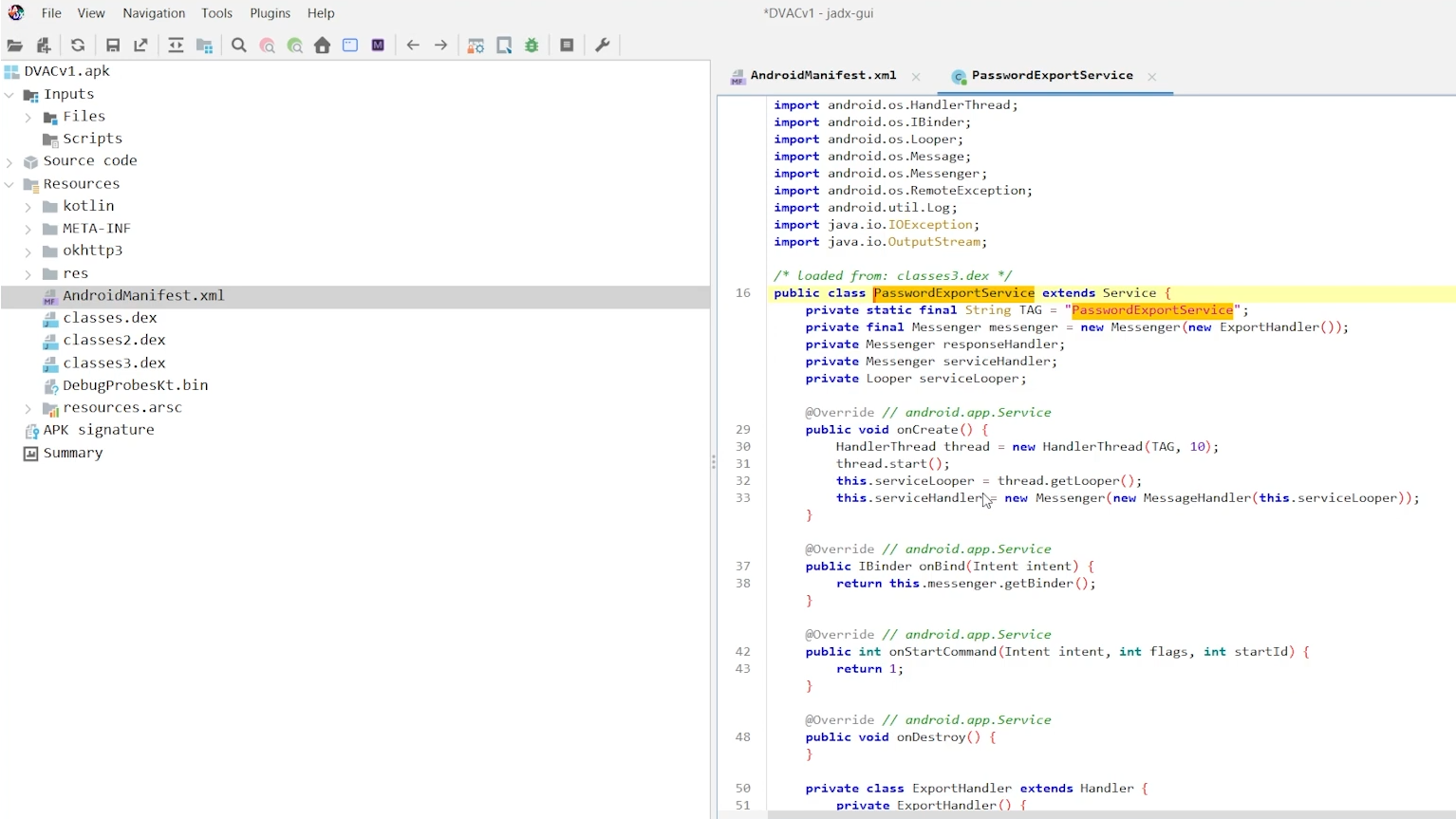
* **Preparation**:
  + Set up Android emulator or physical device with USB debugging enabled.
  + Install tools: MobSF (via Docker or local setup), Frida (client and server), Drozer (via pip).
* **Static Analysis**:
  + Run MobSF for static analysis: Upload APK, review permissions, code analysis, and malware checks.
  + Document vulnerabilities in a log table (e.g., Test ID, Vulnerability, Severity, Target App).
* **Dynamic Instrumentation**:
  + Hook functions with Frida: Write and inject scripts to monitor/intercept methods.
  + Test for bypasses (e.g., auth, SSL pinning) and log runtime behaviors.
* **IPC Testing**:
  + Test IPC with Drozer: Connect to device (drozer console connect), scan for exposed components (run app.package.attacksurface), and attempt exploits (e.g., run app.activity.start --component com.example.testapp com.example.VulnerableActivity).
  + Verify content providers for data leaks and broadcast receivers for injection risks.

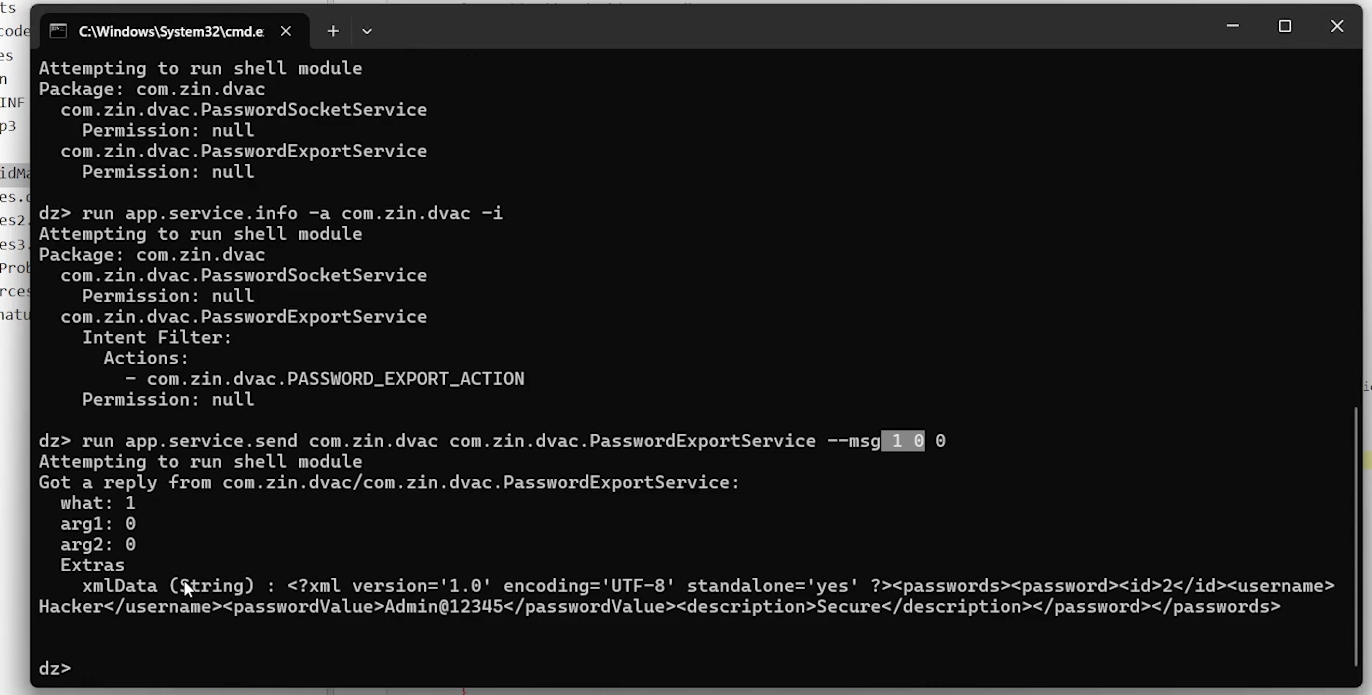












**Post-Testing**:

* + Summarize findings in reports.
  + Clean up: Uninstall apps, revoke permissions, and reset emulator.
  + Review for false positives and update checklist as needed.

This checklist ensures consistency, reduces errors, and serves as a training resource.

**Overall Findings and Recommendations**

* **Key Vulnerabilities Identified**: Insecure storage (high severity) and authentication bypass (critical), common in poorly secured apps.
* **Tool Effectiveness**: MobSF provided quick static insights, Frida enabled precise dynamic exploits, and Drozer (though not detailed in enhanced tasks) complemented by targeting IPC.
* **Risk Assessment**: The test.apk scored poorly on security, with potential for real-world exploitation leading to data loss or compliance violations (e.g., GDPR).
* **General Recommendations**:
  + Adopt secure coding practices from the start, including encryption and input validation.
  + Integrate automated testing in CI/CD pipelines using these tools.
  + Conduct regular pentests and update apps based on evolving threats.
* **Limitations**: Lab focused on Android; iOS testing (e.g., with Objection) could expand scope. Emulated environments may not capture all hardware-specific issues.

**Conclusion**

This lab demonstrated the power of MobSF, Frida, and Drozer in uncovering and exploiting mobile vulnerabilities. By documenting findings and creating checklists, the exercise promotes proactive security. Future iterations could include iOS apps or advanced scenarios like reverse engineering obfuscated code.