Static Analysis of API Usages in Java Classes

Abstract

This report documents the development and implementation of a static analysis tool using the Soot framework. The tool aims to reveal API usage in Java classes, specifically those compiled with Android libraries. Understanding API usage is crucial for identifying potential security vulnerabilities, such as capability leakage and data exfiltration, that arise from misusing sensitive functionalities like GPS location and SMS services.

Introduction

The proliferation of Android applications necessitates robust security measures to prevent misuse of sensitive functionalities. Static analysis offers a preemptive approach to identifying misuses by analyzing the code without execution. This project focuses on developing a custom analyzer with Soot to study API usage in compiled Java classes, aiding in identifying potential security risks.

Objectives

- To implement a static analysis tool capable of identifying API usage in Java classes compiled with Android libraries.
- To leverage the Soot framework for analyzing and transforming Java bytecode to achieve the project's goal.
- To facilitate the detection of potential security vulnerabilities in Android applications through automated static analysis.

Methodology

Environment Setup

- Installation of Java and configuration of the Soot framework.
- As I used Java 11, I used the provided *sootclasses_j9-trunk-jar-with-dependencies.jar* file
- Preparation of lab2.zip, containing the analyzer template and target Java class (Unknown.class).

Implementation

Compilation

Compiled the analyzer using Java Compiler (javac), specifying the classpath for Soot and Android libraries.

```
javac -cp sootclasses_j9-trunk-jar-with-dependencies.jar:. analyzer/SimpleAPILookupTransformer.java
javac -cp "sootclasses_j9-trunk-jar-with-dependencies.jar:." analyzer/AnalysisMain.java
```

Execution

Executed the analyzer to process the target Java class and reveal API usage.

```
java -cp sootclasses_j9-trunk-jar-with-dependencies.jar:android-17.jar:. analyzer.AnalysisMain ./unknown
```

Development of SimpleAPILookupTransformer

- I Extended BodyTransformer to analyze method bodies and record API invocations.
- I Implemented logic to distinguish between API calls and internal method calls by checking the declaring class of invoked methods.
    - Iteration over Statements.
    - Check for Invocation Expressions
    - Retrieve the Invocation Expression
    - Get the Method and Declaring Class
    - Check if the Method is Part of an API
    - Record the API Signature

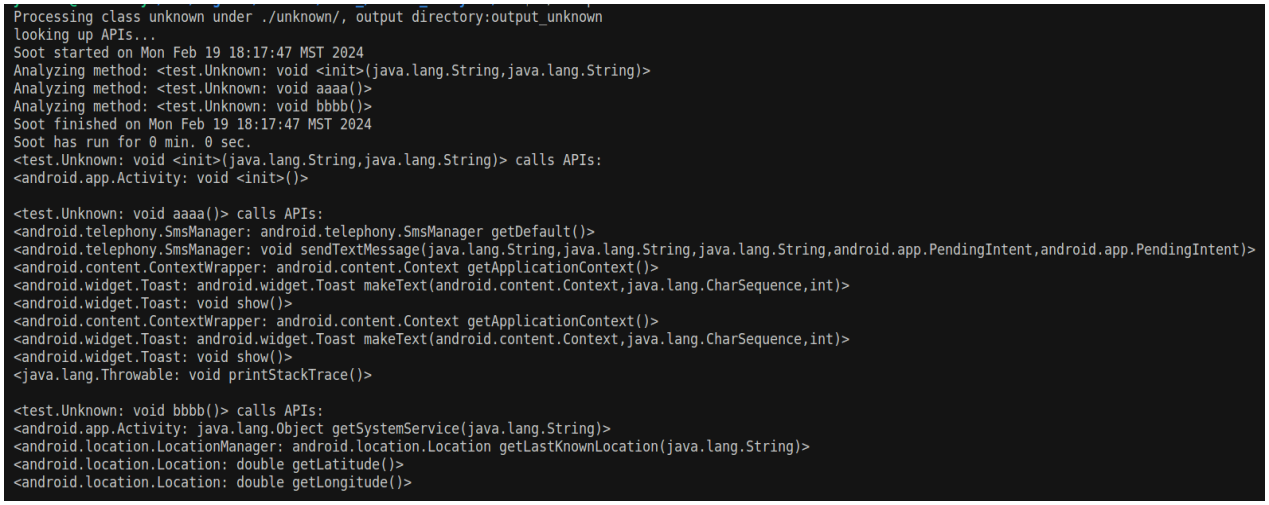
```
    Stmt s = (Stmt)iter.next();

    if(s.containsInvokeExpr()){
        InvokeExpr invokeExpr = s.getInvokeExpr();
        SootMethod invSootMethod = invokeExpr.getMethod();
        SootClass  declaringSootClass = invSootMethod.getDeclaringClass();

        if(!declaringSootClass.isApplicationClass()){
            String apiSignature = invSootMethod.getSignature();
            apis.add(apiSignature);
        }
    }
```

Result & Analysis

The analysis successfully identified API usages within the *Unknown.class*, revealing calls to sensitive Android APIs. The output directory, named after the processed class, contained detailed information about the APIs invoked by each method, offering insights into potential security vulnerabilities.



```
Processing class unknown under ./unknown/, output directory:output_unknown
looking up APIs...
Soot started on Mon Feb 19 18:17:47 MST 2024
Analyzing method: <test.Unknown: void <init>(java.lang.String,java.lang.String)>
Analyzing method: <test.Unknown: void aaaa()>
Analyzing method: <test.Unknown: void bbbb()>
Soot finished on Mon Feb 19 18:17:47 MST 2024
Soot has run for 0 min. 0 sec.
<test.Unknown: void <init>(java.lang.String,java.lang.String)> calls APIs:
<android.app.Activity: void <init>()>

<test.Unknown: void aaaa()> calls APIs:
<android.telephony.SmsManager: android.telephony.SmsManager getDefault()>
<android.telephony.SmsManager: void sendTextMessage(java.lang.String,java.lang.String,java.lang.String,android.app.PendingIntent,android.app.PendingIntent)>
<android.content.ContextWrapper: android.content.Context getApplicationContext()>
<android.widget.Toast: android.widget.Toast makeText(android.content.Context,java.lang.CharSequence,int)>
<android.widget.Toast: void show()>
<android.content.ContextWrapper: android.content.Context getApplicationContext()>
<android.widget.Toast: android.widget.Toast makeText(android.content.Context,java.lang.CharSequence,int)>
<android.widget.Toast: void show()>
<java.lang.Throwable: void printStackTrace()>

<test.Unknown: void bbbb()> calls APIs:
<android.app.Activity: java.lang.Object getSystemService(java.lang.String)>
<android.location.LocationManager: android.location.Location getLastKnownLocation(java.lang.String)>
<android.location.Location: double getLatitude()>
<android.location.Location: double getLongitude()>
```

Fig: The final analysis output

The output shows that the *test.Unknown* class is likely involved in functionalities that require sending SMS messages and accessing the device's GPS location—features that can be sensitive due to privacy concerns and potential for misuse. API calls to SmsManager and LocationManager within the activity indicate that this class handles communication and location-based services. The explicit printing of stack traces and user notifications via Toast messages also suggests an attempt to manage errors and inform the user of actions being performed.

From a security perspective, the use of such APIs should be scrutinized to ensure they adhere to best practices for permission handling, user consent, and data protection. The static analysis results would prompt a security reviewer to check for appropriate permission requests in the application manifest and validate that the application's runtime behavior includes proper user notifications and consent dialogs before accessing SMS and location services.

Discussion

The tool's ability to reveal API usage highlights its potential to identify security risks in Android applications. Future enhancements could include deeper analysis to detect misuse patterns and integration into continuous integration pipelines for automated security audits.

Conclusion

Developing a static analysis tool with Soot demonstrates a practical approach to identifying potential security vulnerabilities in Android apps through API usage analysis. This tool is a foundation for further research and development in application security.