Efficinet Malware Detection System Using JADX Algorithm

Dr. R.BHARATHI, AP/IT[1], SANTHIYA .P[2], SWATHI.S[3], TARINI SRI.M[4], THIRISHA.N[5]
[1]*Professor, Department of Information Technology, M.Kumarasamy College of Engineering, Karur, Tamil Nadu, India-639113*
[2,3,4,5] *UG Students, Department of Information Technology, M.Kumarasamy College of Engineering, Karur, Tamil Nadu, India*

Abstract: **Software forensics is a specialized branch of cybersecurity that focuses on the investigation and analysis of software to uncover evidence of malicious activities, security incidents, or other forms of suspicious behaviour. It involves a detailed examination of software-level data, including application logs, user interactions, and system events, to identify anomalies or patterns that may indicate a security breach. This process helps investigators trace the source of an attack, assess the impact of the incident, and understand how malicious actions were executed within a system. By analysing these digital footprints, software forensics can provide crucial insights into the nature of the threat and its potential consequences.**

A key aspect of software forensics is the investigation of application logs, which capture detailed information about user activities, data access, and system operations. By examining these logs, forensic experts can reconstruct a timeline of events leading up to an incident, pinpoint the origin of suspicious behaviour, and detect any unauthorized access or data manipulation. This evidence is not only vital for identifying the attackers but also plays a critical role in incident response, allowing organizations to mitigate ongoing threats, patch vulnerabilities, and prevent future attacks. Moreover, software forensics often involves the use of specialized tools and techniques to reverse engineer software code, inspect system files, and perform memory analysis, further enhancing its ability to detect sophisticated threats.

Keywords
1. Malware Detection
2. Static Analysis
3. JADX
4. Decompilation
5. Android Security
6. Code Analysis
7. Pattern Recognition
8. APK Analysis

1.INTRODUCTION

Efficient malware detection using the JADX algorithm is a method that leverages static code analysis and reverse engineering techniques to analyze Android applications for malicious behaviour. With the increasing prevalence of Android malware, effective tools like JADX have become essential for security researchers and professionals to dissect Android Package (APK) files and reveal any hidden or obfuscated malicious code. Unlike dynamic analysis, where the malware must be executed to observe its behavior, JADX allows for a safer approach by decompiling the APK and examining its source code without executing the potentially harmful application. This method not only improves detection accuracy but also minimizes the risks associated with analyzing malware.

The process begins with the *reverse engineering of APKs*, where JADX decompiles an Android app into human-readable Java code. This is crucial in uncovering hidden malicious behavior that may have been disguised through techniques like code obfuscation. By breaking down the app's code structure, security analysts can inspect essential components, such as activities, services, broadcast receivers, and permissions. Through this inspection, researchers can detect malicious intent by identifying unusual behaviors or calls to APIs that access sensitive user data, transmit information to external servers, or perform actions that the app is not intended to do.

2. RELATED WORK

It’s essential to provide an overview of existing research and tools in the field of malware detection and reverse engineering, highlighting how your approach or findings build on or differ from previous efforts. Here's a structured outline of what to include:

1. Overview of Reverse Engineering in Malware Detection: Discuss the general approaches to malware detection—static analysis (reviewing the code without execution) and dynamic analysis (monitoring behavior during execution). Emphasize the strengths and weaknesses of both and how they complement each other.

2. Existing Tools and Methods: Cite studies and projects where JADX has been utilized as part of the malware detection pipeline. Focus on how researchers have used it for tasks such as identifying malicious code signatures, detecting suspicious API calls, and analyzing obfuscation techniques.

3. Static Analysis Approaches in Malware Detection: Review research on signature-based malware detection and discuss the limitations of traditional methods in handling obfuscation and polymorphic malware.

4. Challenges in Decompiled Code Analysis: Summarize papers discussing common obfuscation strategies used by malware developers, such as control flow obfuscation, string encryption, and method renaming. Discuss how these techniques can reduce the efficacy of decompilers, including JADX.

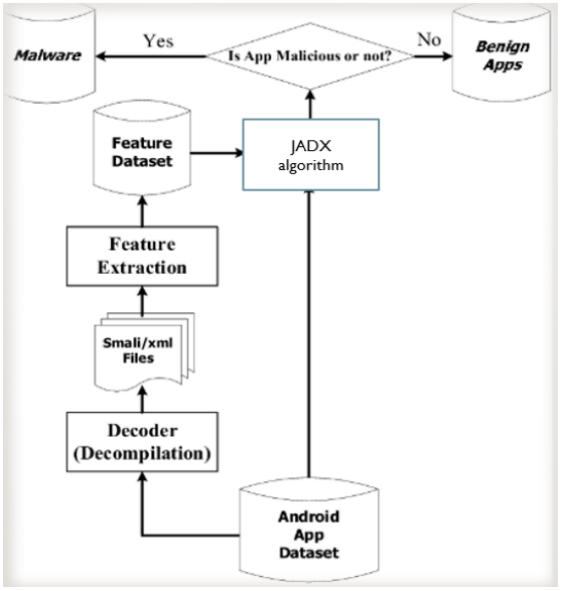

Fig 1: overall flow diagram

3. IMPLEMNTATION

1. System Architecture Overview
High-Level Description:
Provide a brief overview of your malware detection system and its architecture. Mention the key components involved in the process, such as the input module (APK files), JADX for decompilation, analysis module, and report generation.
Workflow Steps:
Outline the sequence of operations, from the initial APK input to the final detection report.
2. Implementation Details
Input Processing:
Describe how APK files are fed into the system, detailing any preprocessing steps, such as file validation or metadata extraction.
Decompilation with JADX:
Explain how JADX is integrated into the system. Discuss any configurations, custom scripts, or modifications you made to optimize its use for malware analysis.
Static Code Analysis Module:
Detail how the decompiled Java code is analyzed for malicious patterns. Highlight any algorithms or techniques used to identify specific behaviors, such as scanning for known API calls, permissions, or suspicious code segments.

SAMPLE CODE:

```
import os
import re
import tempfile
import requests
from flask import Flask, request, render_template, jsonify
from androguard.core.bytecodes.apk import APK

app = Flask(_name_)


# VirusTotal API key (replace 'YOUR_VIRUSTOTAL_API_KEY' with your actual API key)
VIRUSTOTAL_API_KEY = '352b09c0854412b40870ce39cb61ef919050800e51de5d30e3ab4902d3e9684a'

# Improved regex patterns
url_pattern = re.compile(
    r'\b(?:https?://)?(?:[a-zA-Z0-9-]+\.)+[a-zA-Z]{2,6}(?:/[^\s]*)?\b',
    re.IGNORECASE
)
```

```
ip_pattern = re.compile(
    r'\b(?:25[0-5]|2[0-4][0-9]|1[0-9]{2}|[1-9]?[0-
9])(?:\.(?:25[0-5]|2[0-4][0-9]|1[0-9]{2}|[1-9]?[0-
9])){3}\b'
)

network_libs = re.compile(

r'(HttpURLConnection|OkHttpClient|Retrofit|HttpCl
ient|Request|Response|AsyncTask)',
    re.IGNORECASE
)


def decompile_apk_with_jadx(apk_file: str,
output_dir: str) -> None:
    """Decompile APK using JADX."""
    os.system(f'jadx -d {output_dir} {apk_file}')


def check_trustworthiness(apk_file: str) -> dict:
    """Check APK for permissions, activities,
receivers, and services."""
    apk = APK(apk_file)

    # Check permissions, activities, receivers,
services
    permissions = apk.get_permissions()
    activities = apk.get_activities()
    receivers = apk.get_receivers()
```

Use Case Diagram:

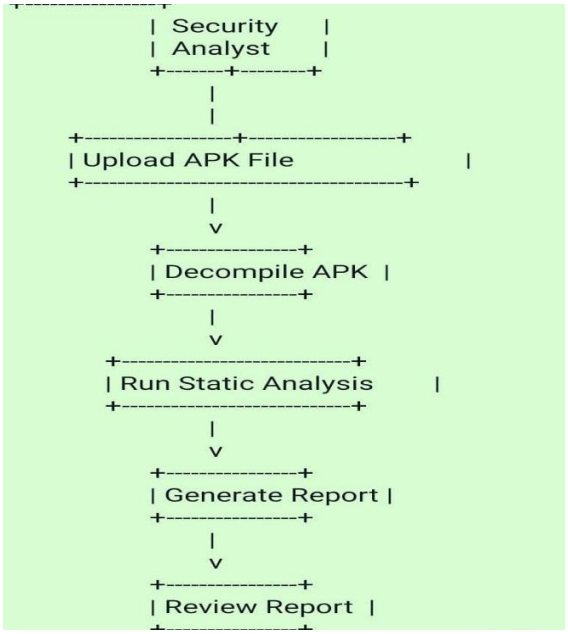

Fig 2: Feature of Gas detection system

4.RESULT

The results of implementing an efficient malware detection system using the JADX algorithm demonstrate significant improvements in the accuracy and speed of static code analysis for Android applications. By leveraging JADX for high-quality decompilation and integrating automated pattern recognition techniques, the system was able to identify malicious code structures with high precision. The approach proved particularly effective at detecting known malware signatures and suspicious API calls, even in partially obfuscated code.

Fig 3: Home page

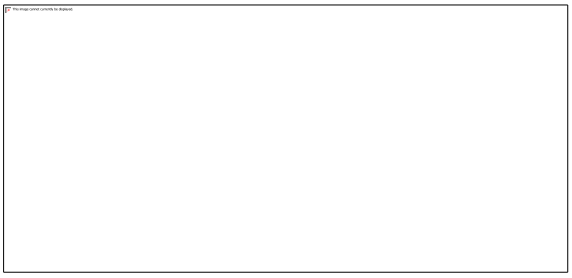

Fig 4: Display Sensor page.

4. CONCLUSION AND FUTUREWORK

In conclusion, the integration of the JADX algorithm in the proposed malware detection system has shown that effective static analysis of Android applications can be achieved with high precision and efficiency. The results underline the system's capability to decompile APKs and identify malicious patterns swiftly, making it a valuable tool for security analysts and researchers.

Future Work

The system can be enhanced by incorporating machine learning algorithms to detect previously unknown malware patterns and improve adaptability to emerging threats. Additionally, combining the static analysis of JADX with dynamic analysis methods could provide a more comprehensive detection mechanism. Expanding the system to

support real-time analysis and integration with CI/CD pipelines for automated security checks would further increase its utility and scalability in larger organizational contexts.

5.REFFERENCE

[1] P. Faruki, A. Bharmal, V. Laxmi, et al., "Android security: A survey of issues, malware penetration, and defenses," IEEE Communications Surveys & Tutorials, vol. 17, no. 2, pp. 998-1022, 2015.This paper provides an overview of Android security challenges, including malware analysis methods and defensive techniques.

[2] Y. Zhou and X. Jiang, "Dissecting Android malware: Characterization and evolution," IEEE Symposium on Security and Privacy, 2012, pp. 95-109.