Analysis of Android Malware Detection Using DREBIN API Features

Exploring the Efficacy of API Features in Classifying Android Malware

Project Overview

Introduction to Static Analysis Shortcomings:
Despite its efficiency, static analysis alone falls short in detecting sophisticated malware that employs dynamic execution and obfuscation techniques.

The Promise of API Analysis:
By examining the patterns of API usage that differentiate malware from benign applications, we aim to uncover the hidden behaviors of sophisticated malware.

Research Goal

- Enhance the capabilities of static malware analysis

- Integrate API usage pattern analysis, covering gaps left by static analysis alone

Problems and Objectives

The Core Issue:
Static analysis's inability to detect malware that dynamically loads code or uses obfuscation techniques.

Aim of the Study:
To develop a nuanced understanding of malware behavior through the lens of API call sequences, enriching static analysis with dynamic insights.

Problems and Objectives

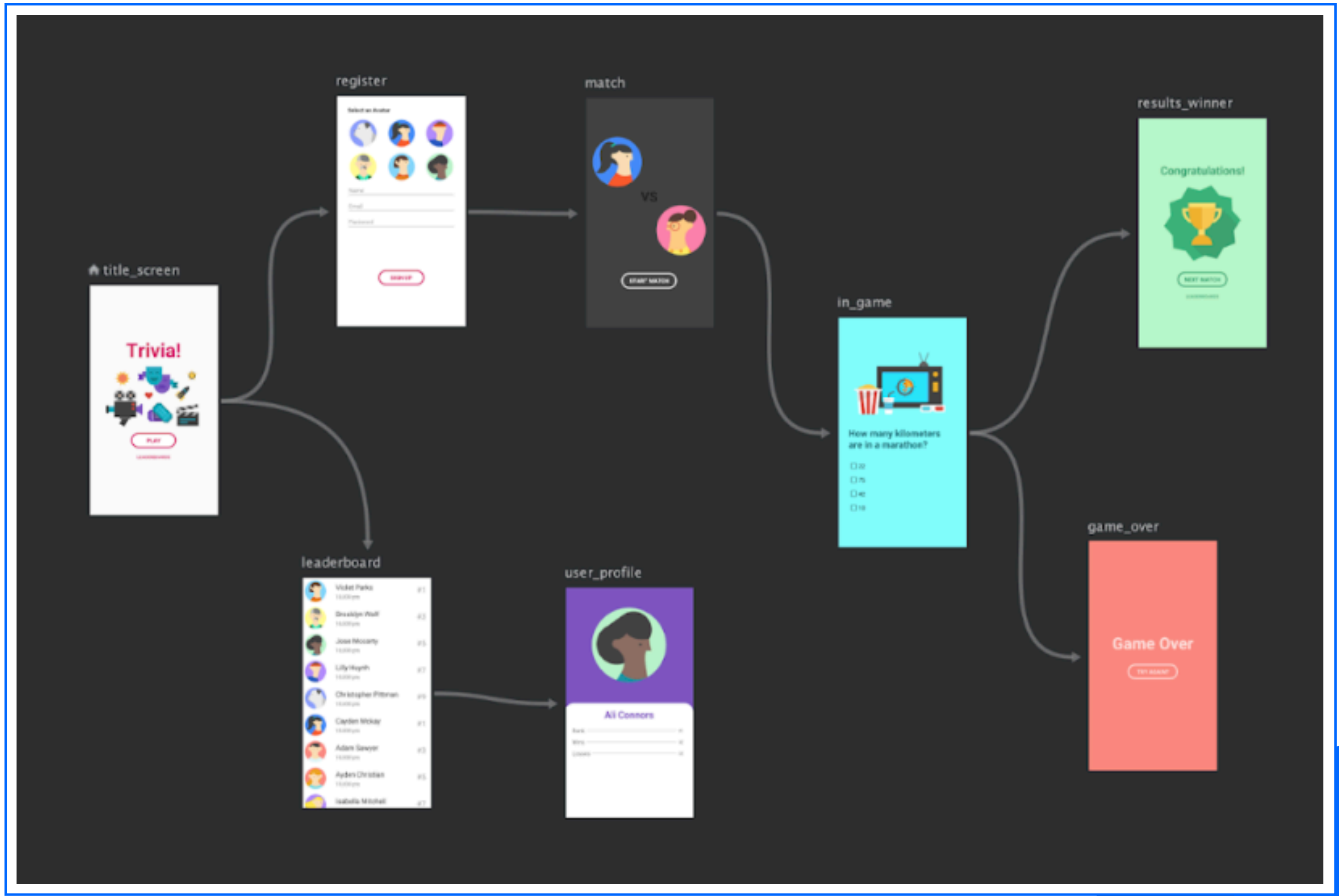
Key Objectives:
- Cataloguing Android API calls frequently manipulated by malware
- Designing a classifier capable of discerning malware from benign software based on API call patterns.
- Evaluating the performance of different machine learning models in recognising these patterns effectively.

Equipment and Tools

Java and Assembly Languages:
Essential for dissecting malware code and understanding its construction.

Tool - Android Studio:
Our primary toolkit for developing, testing, and analysing Android applications.

Equipment and Tools

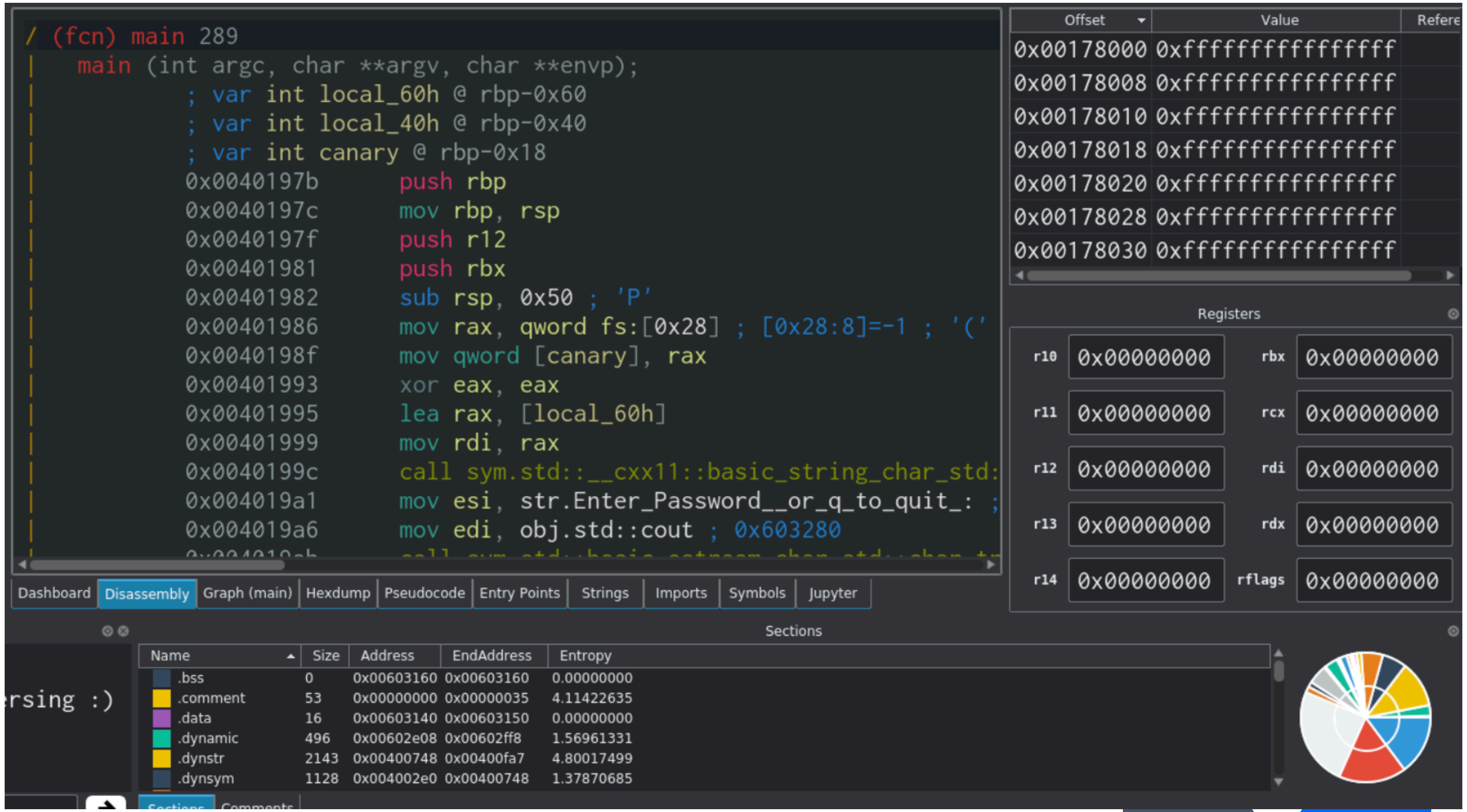


Equipment and Tools

Cutter:
A reverse engineering platform used for decompiling malware binaries, providing insights into their operational logic.

TensorFlow and PyTorch:
Leading machine learning frameworks selected for model development, offering robust tools for analysing API usage patterns.

Equipment and Tools



Expected Challenges

Distinguishing Malicious from Legitimate API Use:
Address the complexity of identifying API calls that are malicious in context but might appear in legitimate applications.

Dealing with Code Obfuscation:
Anticipate difficulties in analyzing malware that employs sophisticated obfuscation techniques, affecting the visibility of API call sequences.

Future work

Incorporating Dynamic Analysis:
Explore plans to integrate dynamic analysis for a more holistic view of malware behavior, potentially automating the transition from static to dynamic analysis based on specific API pattern triggers.

Advanced Machine Learning Techniques:
Consider the exploration of more sophisticated AI approaches, including neural networks or unsupervised learning models, to improve detection rates and reduce false positives.

Thank you