
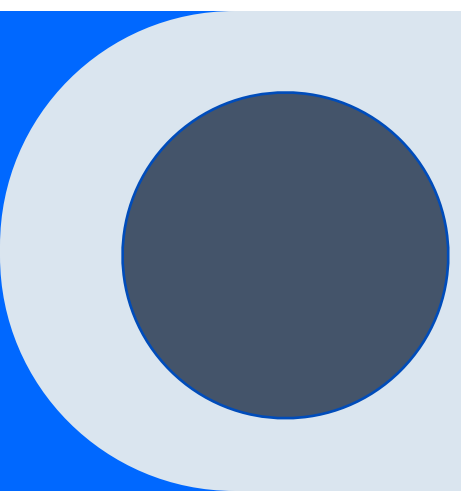




Analysis of Android Malware Detection Using DREBIN API Features

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

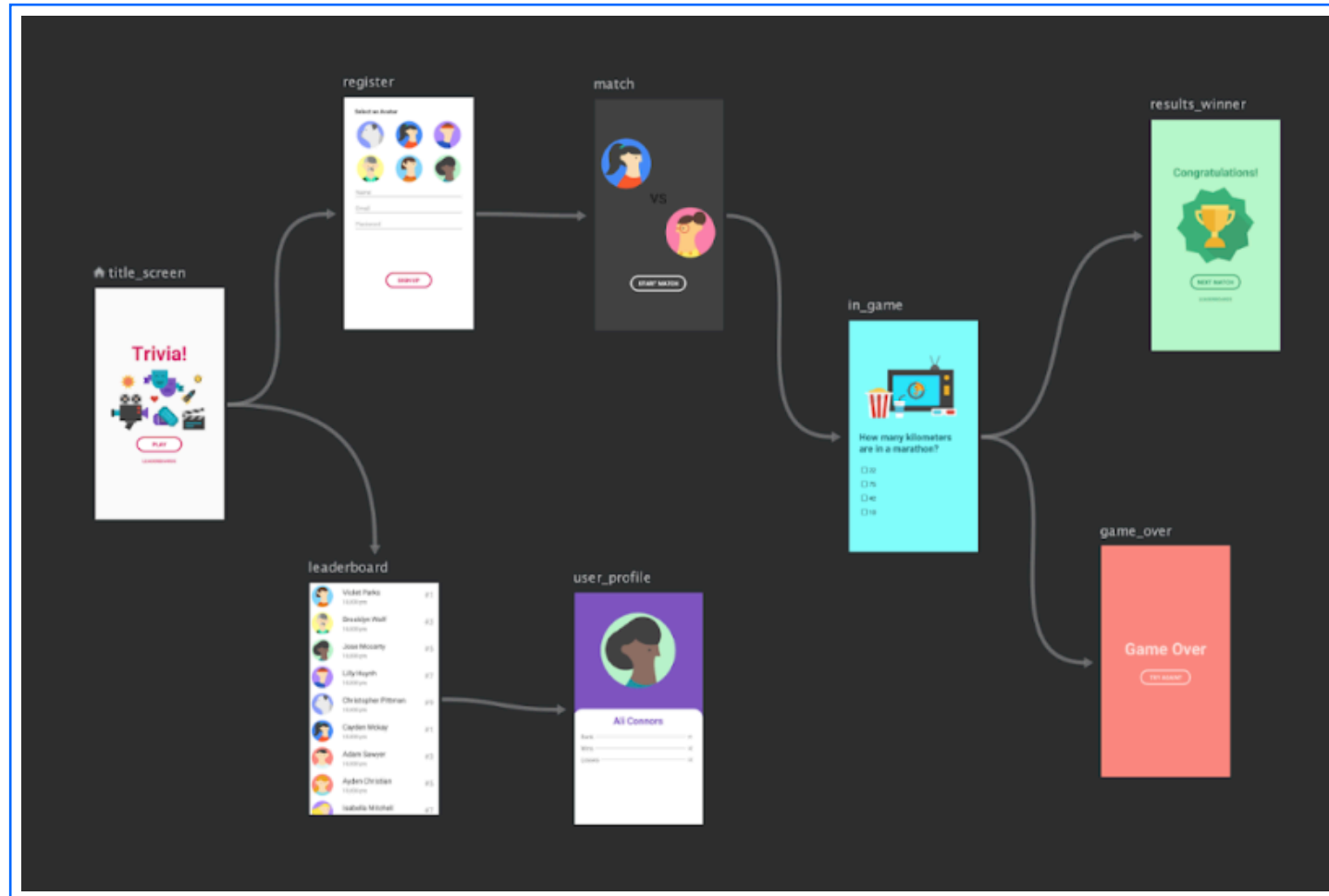
Java and Assembly Languages:

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

The screenshot displays a debugger interface with three main panels. The left panel shows assembly code for a function named `main`. The middle panel shows the state of CPU registers. The right panel shows a list of memory sections.

Assembly Code:

```
(fcn) main 289
main (int argc, char **argv, char **envp);
; var int local_60h @ rbp-0x60
; var int local_40h @ rbp-0x40
; var int canary @ rbp-0x18
0x0040197b push rbp
0x0040197c mov rbp, rsp
0x0040197f push r12
0x00401981 push rbx
0x00401982 sub rsp, 0x50 ; 'P'
0x00401986 mov rax, qword fs:[0x28] ; [0x28:8]=-1 ; '('
0x0040198f mov qword [canary], rax
0x00401993 xor eax, eax
0x00401995 lea rax, [local_60h]
0x00401999 mov rdi, rax
0x0040199c call sym.std::__cxx11::basic_string_char_std:
0x004019a1 mov esi, str.Enter_Password__or_q_to_quit_ ;
0x004019a6 mov edi, obj.std::cout ; 0x603280
0x004019ab call sym.std::basic_string_char_std::operator<>
```

Registers:

Register	Value
r10	0x00000000
r11	0x00000000
r12	0x00000000
r13	0x00000000
r14	0x00000000
rbx	0x00000000
rcx	0x00000000
rdi	0x00000000
rdx	0x00000000
rflags	0x00000000

Sections:

Name	Size	Address	EndAddress	Entropy
.bss	0	0x00603160	0x00603160	0.00000000
.comment	53	0x00000000	0x00000035	4.11422635
.data	16	0x00603140	0x00603150	0.00000000
.dynamic	496	0x00602e08	0x00602ff8	1.56961331
.dynstr	2143	0x00400748	0x00400fa7	4.80017499
.dynsym	1128	0x004002e0	0x00400748	1.37870685

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