

# Sistem de Clasificare Malware Android

Lucrare de Licență

**Crețu Vlad-Sebastian**

Coordonator: Dr. Ing. Aciu Răzvan-Mihai



Universitatea Politehnica Timișoara  
Facultatea de Automatică și Calculatoare  
2024

# Cuprins

<b>1</b>	<b>Introducere</b>	<b>3</b>
<b>2</b>	<b>Analiza stadiului actual în domeniul problemei</b>	<b>4</b>
2.1	VirusTotal . . . . .	4
2.2	Hybrid Analysis . . . . .	4
2.3	Jotti's Malware Scan . . . . .	5
2.4	Comparație . . . . .	5
<b>3</b>	<b>Fundamentare teoretică</b>	<b>7</b>
3.1	Android . . . . .	7
3.1.1	Aplicații . . . . .	7
3.1.2	Permisuni . . . . .	8
3.1.3	Fișierul manifest . . . . .	8
3.1.4	Android Runtime și Dalvik . . . . .	8
3.2	Rust . . . . .	8
3.3	Tensorflow . . . . .	9
3.4	Axum . . . . .	9
3.5	React . . . . .	9
3.6	MongoDB . . . . .	10
<b>4</b>	<b>Soluția propusă</b>	<b>11</b>
<b>5</b>	<b>Implementare</b>	<b>12</b>
5.1	Frontend . . . . .	12
5.2	REST API . . . . .	12
5.2.1	Scanare . . . . .	13
5.3	Extragere de date . . . . .	13
5.3.1	Componente și permisiuni . . . . .	15
5.3.2	Bytecode . . . . .	15
5.3.3	Preprocesare . . . . .	16
5.3.4	Testare . . . . .	19
5.4	Clasificare . . . . .	21
5.4.1	Dataset . . . . .	21
5.4.2	Model . . . . .	22
5.4.3	Nivel local . . . . .	22
5.4.4	Nivel de metodă . . . . .	23
5.4.5	Nivel global . . . . .	23
5.4.6	Permisuni . . . . .	24
5.4.7	Output . . . . .	24

5.4.8	Antrenare . . . . .	25
<b>6</b>	<b>Rezultate experimentale</b>	<b>27</b>
6.1	Descrierea procesului de testare . . . . .	27
6.2	Rezultate experimentale . . . . .	27
<b>7</b>	<b>Concluzii, contribuții și direcții de continuare a dezvoltării</b>	<b>28</b>
7.1	Concluzii . . . . .	28
7.2	Contribuții . . . . .	28
7.3	Direcții viitoare . . . . .	28
	<b>Bibliografie</b>	<b>28</b>

# 1. Introducere

Lucrarea se concentrează pe dezvoltarea unui sistem de clasificare multi-label a fișierelor APK<sup>29</sup> folosind 5 componente principale:

- Bibliotecă de preprocesare a datelor realizată folosind limbajul de programare Rust<sup>26</sup>, care se ocupă de extragerea permisiunilor și secvențelor de cod din fișierele APK.
- Model machine-learning de clasificare realizat folosind biblioteca Tensorflow<sup>1</sup>, care va clasifica fișierele APK în una din 5 categorii: *Adware*, *Banking*, *Benign*, *Riskware*, *SMS*.
- REST API realizat folosind framework-ul axum<sup>27</sup>, care va oferi o interfață programatică pentru a trimite fișiere APK spre clasificare și a primi rezultatele.
- Bază de date MongoDB<sup>10</sup> pentru stocarea/interogarea rezultatelor clasificării.
- Aplicație web pentru interfațare manuală cu API-ul și vizualizarea predicțiilor, realizată folosind framework-ul React<sup>28</sup>.

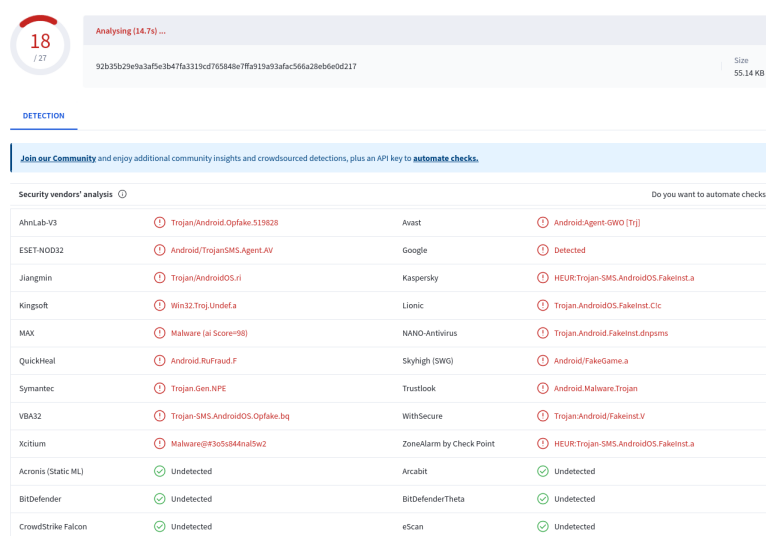
Utilizatorul final va putea trimite:

- Unul sau mai multe fișiere APK către sistem și va primi înapoi predicțiile pentru fiecare fișier.
- Un hash SHA256 al unui fișier APK și va primi înapoi predicția pentru acel fișier, cu condiția că fișierul a fost scanat anterior și salvat în baza de date.

## 2. Analiza stadiului actual în domeniul problemei

### 2.1 VirusTotal

VirusTotal<sup>25</sup> este un serviciu web care analizează fișiere și URL-uri cu peste 70 de servicii de scanare third-party, tehnici de sandboxing și unelte de pattern-matching. Orice utilizator poate selecta un fișier sau un URL de pe dispozitivul său și să-l trimită la VirusTotal pentru investigație.



Analysing (14.7s) ...		Size: 55.14 KB	
92b35b22e9a3af5c3b47fa3313cd7f05849e7f9e119a33afac566a28eb6ed0217			
DETECTION			
Join our Community and enjoy additional community insights and crowdsourced detections, plus an API key to automate checks.			
Security vendors' analysis		Do you want to automate checks?	
AhnLab-V3	Trojan.Android.Opfake.519828	Avast	Android.Agent-GWO [Trj]
ESET-NOD32	Android/TrojanSMS.Agent.AV	Google	Detected
Jiangmin	Trojan.AndroidOS.ri	Kaspersky	HEUR:Trojan-SMS.AndroidOS.FakeInst.a
Kingsoft	Win32.Troj.Undef.a	Lionic	Trojan.AndroidOS.FakeInst.Cic
MAX	Malware (ai Score=98)	NANO-Antivirus	Trojan.Android.FakeInst.dnpsms
QuickHeal	Android.RuFraud.F	Skyhigh (SWG)	Android/FakeGame.a
Symantec	Trojan.Gen.NPE	Trustlook	Android.Malware.Trojan
VBA32	Trojan-SMS.AndroidOS.Opfake.bq	WithSecure	Trojan.Android/FakeInst.V
Xcitium	Malware@#3o5s844nal5w2	ZoneAlarm by Check Point	HEUR:Trojan-SMS.AndroidOS.FakeInst.a
Acronis (Static ML)	Undetected	Arcabit	Undetected
BitDefender	Undetected	BitDefenderTheta	Undetected
CrowdStrike Falcon	Undetected	eScan	Undetected

Figura 2.1: Procesul de scanare a unui fișier cu VirusTotal

### 2.2 Hybrid Analysis

Hybrid Analysis<sup>2</sup> este un serviciu gratuit de analiză malware. Utilizând acest serviciu, utilizatorii pot trimite fișiere pentru o analiză statică și dinamică detaliată. Sistemul de analiză implicit suportă fișiere executabile, office, pdf, apk, jar și altele. Acesta folosește analiză statică, analiză dinamică prin sandboxing și rezultatele antivirusului MetaDefender<sup>24</sup>.

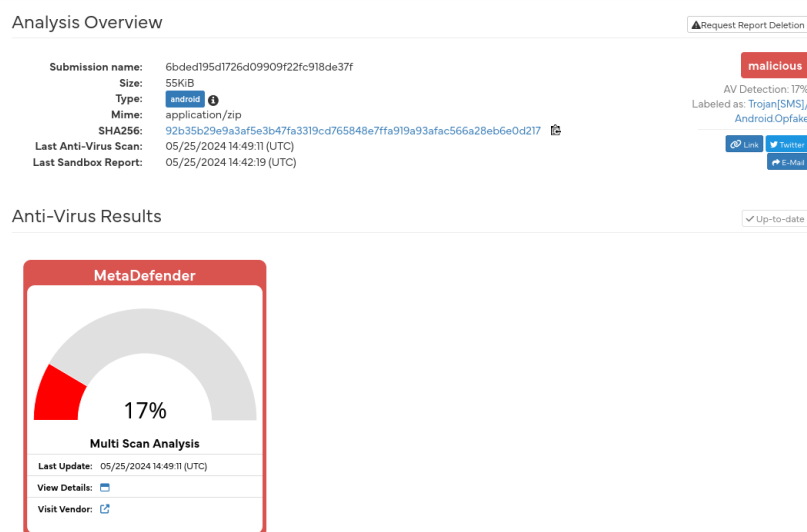


Figura 2.2: Rezultatele analizei unui fișier cu Hybrid Analysis

## 2.3 Jotti's Malware Scan

Jotti's Malware Scan<sup>12</sup> este un serviciu similar celor de mai sus, care permite scanarea folosind produsele a 14 vendori populari de antivirusi, și căutarea rezultatelor scanării fișierelor după hash.

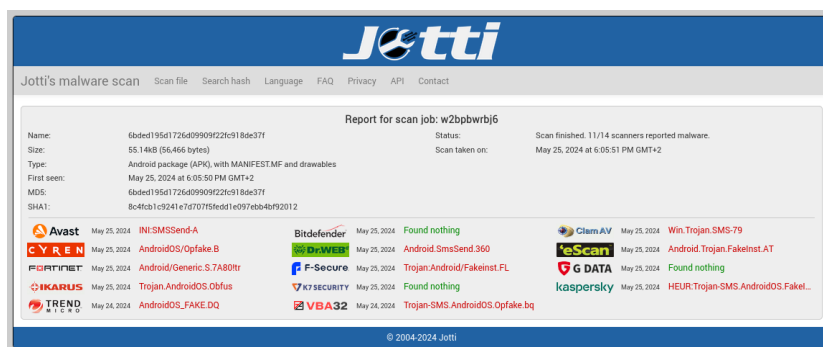


Figura 2.3: Rezultatele analizei unui fișier cu Jotti's Malware Scan

## 2.4 Comparație

În tabela 2.1 sunt comparate caracteristicile de analiză statică și căutare a serviciilor menționate anterior. Pentru referință vor fi folosite următoarele:

1. **Scanare proprie** - Algoritm propriu pentru analiză statică a fișierelor APK și nu doar grupare a rezultatelor vendorilor third-party.
2. **Durată** - Durata medie de analiză a unui fișier APK în secunde. Metrica a fost măsurată folosind 5 fișiere scanate consecutiv de 3 ori.
3. **Căutare hash** - Căutare a rezultatelor scanării unui fișier după hash.
4. **Căutare conținut** - Oferă posibilitatea interogării dacă o secvență de octeți se consideră malware.

Caracteristică	Serviciul meu	VirusTotal	Hybrid Analysis	Jotti's Malware Scan
Scanare proprie	✓	-	✓	-
Durată	3.3	86.2	62.4	19.1
Căutare hash	✓	✓	✓	✓
Căutare conținut	-	-	✓	-

Tabela 2.1: Compararea caracteristicilor de analiză statică și căutare a serviciilor

## 3. Fundamentare teoretică

### 3.1 Android

Android este un sistem de operare open source complet cu cod sursă personalizabil care poate fi portat pe aproape orice dispozitiv, și cu documentație publică disponibilă pentru toată lumea la *source.android.com* pentru dispozitive mobile, condus de Google<sup>18</sup>. Platforma Open System Android (AOSP) este un cod sursă Android disponibil public și modificabil.

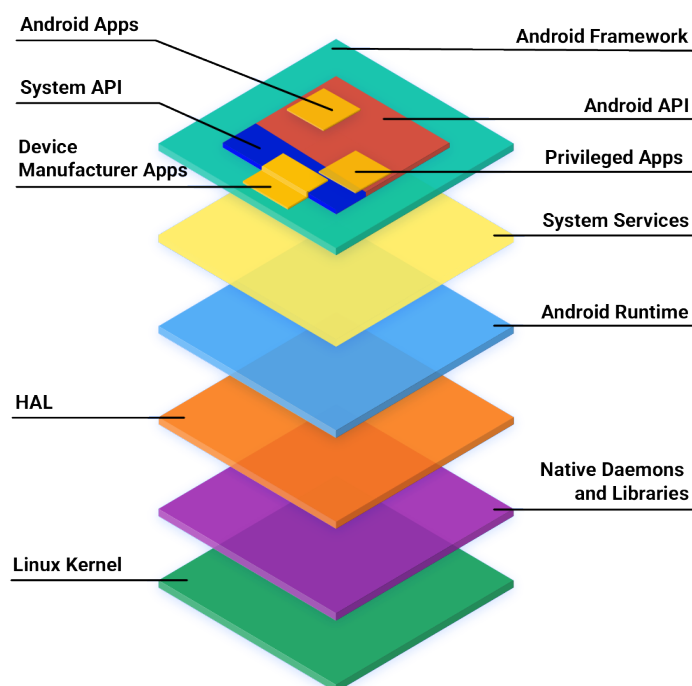


Figura 3.1: AOSP software stack architecture<sup>15</sup>

#### 3.1.1 Aplicații

Aplicațiile Android pot fi scrise folosind Kotlin, limbajul de programare Java și limbajele C++. Instrumentele SDK Android compilează codul împreună cu orice fișiere de date și resurse într-un APK sau un Android App Bundle.<sup>14</sup>

Un pachet Android, care este un fișier de arhivă cu extensia *.apk*, conține fișierele unei aplicații Android necesare la rulare și este fișierul pe care dispozitivele cu Android îl folosesc pentru a instala aplicația.<sup>14</sup>



### 3.1.2 Permisii

Permisunile aplicațiilor ajută la protejarea confidențialității utilizatorilor prin restricționarea accesului la următoarele:

- Date restricționate, cum ar fi starea sistemului și informațiile de contact ale utilizatorilor
- Acțiuni restricționate, cum ar fi conectarea la un dispozitiv împerecheat și înregistrarea audio

Tipul fiecărei permisiuni indică domeniul datelor restricționate la care aplicația poate avea acces și domeniul acțiunilor restricționate pe care aceasta poate efectua atunci când sistemul este acordată.<sup>16</sup>

### 3.1.3 Fișierul manifest

Înainte ca sistemul Android să poată porni o componentă a aplicației, acesta trebuie să știe că acea componentă există citind fișierul manifest al aplicației, *AndroidManifest.xml*. Aplicația declară toate componentele și orice permisiuni de care are nevoie, cum ar fi accesul la internet sau accesul de citire la contactele utilizatorului sale în acest fișier, care se află la rădăcina directorului proiectului aplicației.<sup>14</sup>

### 3.1.4 Android Runtime și Dalvik

Android runtime (ART) este mediul de execuție utilizat de aplicații și unele servicii de sistem pe Android. Acestea execută formatul Dalvik executable (DEX) și specificația bytecode DEX.<sup>17</sup>

Dalvik Executable este un format de fișier binar care conține codul executabil al programului Android. Acesta este bazat pe regiștri și poate fi ușor decompilat pentru a parcurge instrucțiunile fiecărei metode din fiecare clasă a aplicației. Instrucțiunile constituie o secvență între 2 și 10 octeți, fiecare conține câte un opcode (primul octet) și un număr variabil de argumente fie regiștri, constante sau indecși din tabelele de identificatori.

## 3.2 Rust

Rust<sup>26</sup> este un limbaj de programare axat pe siguranță, viteză și concurență, extrem de rapid și eficient din punct de vedere al memoriei care se potrivește pentru servicii unde performanța este un factor critic.

Sistemul de tipuri bogat al limbajului și modelul de ownership garantează siguranța memoriei și a firelor de execuție — permițând eliminarea multor clase de erori în timpul compilării.

### 3.3 Tensorflow

Tensorflow<sup>1</sup> este o bibliotecă open-source de *machine learning* dezvoltată de echipa Google Brain, care oferă un set larg de instrumente high-level pentru construirea, antrenarea și inferența modelelor de inteligență artificială. Acesta este implementat în limbajul C++ și oferă API-uri pentru diferite limbaje cum ar fi Python și Javascript.

### 3.4 Axum

Axum<sup>27</sup> este un framework pentru aplicații web care se concentrează pe ergonomie și modularitate. Acesta permite dezvoltarea rapidă a REST-API-urilor folosind un model de programare asincron, bazat pe următoarele caracteristici cheie:

- Redirecționarea cererilor HTTP către handleri cu un API fără macro-instrucțiuni.
- Parsarea declarativă a cererilor folosind extractori.
- Model simplu și previzibil de gestionare a erorilor.
- Generarea răspunsurilor fără cod redundant.

### 3.5 React

React<sup>28</sup> este o bibliotecă JavaScript pentru construirea interfețelor de utilizator. Avantajele bibliotecii sunt:

- Paradigmă declarativă care face ușoară crearea de UI-uri interactive, iar codul să fie mai predictibil și mai ușor de înțeles.
- Componente reutilizabile care permit definirea unei componente și utilizarea acesteia oriunde în aplicație.

## 3.6 MongoDB

MongoDB<sup>10</sup> este o bază de date NoSQL orientată pe documente cu o serie de avantaje:

- Structura flexibilă a documentelor.
- Interogare simplă a datelor.
- Framework de agregare robust.
- Scalare orizontală.

Documentele sunt structurate într-un format similar JSON numit BSON (Binary JSON), care reprezintă o serializare binară a documentelor, cu anumite extensii care permit utilizarea unor structuri de date mai complexe cum ar fi date calendaristice.

MongoDB este o alegere populară pentru aplicații cu potențial de scalabilitate și volatilitate a datelor.

## 4. Soluția propusă

## 5. Implementare

În acest capitol sunt prezentate detaliile de implementare ale tuturor unităților sistemului.

### 5.1 Frontend

Interfața web este entry-point-ul pentru utilizatorii aplicației. Elaborată folosind framework-ul React<sup>28</sup>, aceasta conține:

- Câmp pentru încărcare fișiere
- Buton upload fișiere
- Câmp pentru introducere hash SHA-256
- Buton pentru verificare hash

După trimiterea fișierelor către backend, interfața va iniția animația unui spinner de încărcare până când nu e primit răspunsul de la server. Răspunsul va fi afișat sub forma unei liste cartonașe care conțin:

- Numele fișierului
- Hash-ul SHA-256
- Categorie prezisă
- Probabilitatea categoriei

Aceeași procedură va fi inițiată în urma trimiterii unui hash, răspunsul de la server fiind similar.

### 5.2 REST API

API-ul backend este realizat folosind framework-ul axum<sup>27</sup>. Este un API asincron care oferă următoarele rute:

- `POST /scan` - Acceptă unul sau mai multe fișiere APK și returnează predicțiile în formatul JSON ca în fig. 5.1
- `GET /query/:hash` - Returnează predicția pentru un fișier APK după hash-ul SHA256, răspunsul e similar cu cel de la `POST /scan` doar că pentru un singur fișier.

```

1 {
2   ... "success": [
3     ... {
4       ... "sha256": "0a7cb27e52927eacabbb7ecc738b0fea50b3967945257c43a67eb753cb465bd0",
5       ... "prediction": {
6         ... "det": "adware",
7         ... "proba": 0.78
8       }
9     },
10    ... {
11      ... "sha256": "d4f92d348254ca39fd4d85400194acce59ae294eab802ef7befd9b5fd2e3d28b",
12      ... "prediction": {
13        ... "det": "benign",
14        ... "proba": 0.99
15      }
16    }
17  ]
18 }

```

Figura 5.1: Exemplu de răspuns al API-ului

### 5.2.1 Scanare

În momentul încărcării fișierelor, se va face o prefiltrare pentru a lăsa doar fișierele al cărui conținut începe cu `b"PK"` (începutul headerului unui fișier ZIP), pentru a preveni încercarea de a scana fișiere ce nu sunt APK-uri.

Înainte de scanare se calculează hash-ul SHA256 și se caută în baza de date MongoDB<sup>10</sup>. Dacă fișierul a fost scanat anterior, se va returna rezultatul salvat în baza de date. În caz contrar se va face predicția pe fișier folosind biblioteca de extragere a datelor și modelul încărcat folosind binding-urile Tensorflow pentru Rust. Rezultatele scanării sunt salvate în baza de date pentru a fi folosite ulterior, iar răspunsul este trimis înapoi către client.

În caz că se încarcă mai multe fișiere odată, procesul va fi optimizat să facă căutare în baza de date cu mai multe hash-uri odată, iar extragerea de date va fi paralelizată folosind biblioteca *rayon*<sup>7</sup> care facilitează conversia calculelor secvențiale în calcule paralele, printr-un API simplu și intuitiv. Scanarea va fi aplicată doar fișierelor care nu au fost găsite în baza de date.

## 5.3 Extragere de date

Biblioteca de extragere a datelor realizată în cadrul lucrării expune funcția:

```
parse<R: Read + Seek>(apk: R) -> Result<Apk, ApkParseError>
```

Aceasta acceptă un buffer de memorie care reprezintă arhiva APK și returnează un rezultat care conține structura de date *Apk* (fig. 5.2) sau o eroare de parsare *ApkParseError*.

```

1 struct Apk {
2     .... manifest: Option<Manifest>,
3     .... methods: Vec<Method>
4 }
5
6 struct Manifest {
7     .... package: Option<String>,
8     .... permissions: HashSet<String>,
9     .... activities: Vec<String>,
10    .... services: Vec<String>,
11    .... receivers: Vec<String>,
12    .... providers: Vec<String>,
13 }
14
15 struct Method {
16     .... signature: Signature,
17     .... insns: Vec<Instruction>,
18 }
19
20 struct Signature {
21     .... class_type: String,
22     .... method_name: String,
23     .... params: Option<Vec<String>>,
24     .... return_type: String,
25 }
26
27 struct Instruction {
28     .... opcode: Opcode, // Valoare u8 al opcode-ului
29     .... method_id: Option<u16>,
30 }

```

Figura 5.2: Structura de date Apk

În momentul apelării, se vor parcurge fișierele din arhivă pentru a găsi:

1. Fișierul *AndroidManifest.xml* al aplicației care declară componentele și permisiunile necesare.
2. Orice fișier binar al cărui header corespunde cu `DEX_FILE_MAGIC`<sup>20</sup>, fiind considerat un fișier Dalvik Executable cu cod al aplicației.

### 5.3.1 Componente și permisiuni

Întrucât fișierul *AndroidManifest.xml* din cadrul arhivei nu este un fișier XML standard, ci o reprezentare serializată binară a documentului, se folosește biblioteca *abxmldecoder*<sup>3</sup> pentru parcurgerea node-urilor și parsarea informației utile.

Componentele și denumirea pachetului sunt salvate pentru a fi folosite ulterior în sortarea metodelor din fișierele DEX. Există 4 tipuri de componente:

- Activități - Interfețe grafice cu care utilizatorul interacționează.
- Servicii - Rulează în fundal și nu au interfață grafică.
- Receivers - Răspund la anumite evenimente cum ar fi primirea unui SMS.
- Providers - Oferă acces la datele aplicației.

Permisiunile sunt extrase din nodurile *uses-permission* și salvate într-un *HashSet<String>*.

### 5.3.2 Bytecode

Un APK poate avea mai multe fișiere DEX, de obicei se găsesc în directorul root al arhivei și sunt denumite *classes.dex*, *classes2.dex*, etc. Dat fiind faptul că aceasta este doar o convenție de denumire, un atacator poate liber ascunde cod executabil în alte directoare sau sub alt nume, de aceea sunt parcurse și verificate cu o expresie regulată `b"6465780A3033[35-39]00"` (Validează versiunile 35-39) header-ele tuturor fișierelor din arhivă.

```
1 def parse_dexes(dexes: list[Dex], regexes: list[Regex] | None) -> list[Method]:
2     graph = {}
3     for dex in dexes:
4         for cls in dex.classes:
5             for mth in cls.methods:
6                 method = Method(mth.code)
7                 graph[method] = method.calls
8     stack = list(graph.keys())
9     if regexes is not None:
10         stack.sort(key=lambda mth: (any(r.match(mth.sig) for r in regexes), mth.sig), reverse=True)
11     else:
12         stack.sort(reverse=True)
13     flattened = []
14     while stack:
15         method = stack.pop()
16         if not method.visited:
17             flattened.append(method)
18             stack.extend(reversed(method.calls))
19             method.visited = True
20     return flattened
```

Figura 5.3: Algoritm de parsare a metodelor (Abstractizat)



Conform algoritmului, sunt parcurse toate metodele din fişierele DEX pentru a se construi un Control Flow Graph. Graph-ul este aplatizat şi sortat în aşa manieră ca metodele din clasele declarate ca componente în Manifest să fie parcurse primele:

1. **Major Order** - Clasa este declarată în AndroidManifest.xml ca componentă
2. **Minor Order** - Signatura metodei, ex. `Lcom/test/MAct; ->v(I;)V`

După această sortare graph-ul este reparcurs folosind Depth-First Search<sup>30</sup> pentru a se construi vectorul de metode normalizat utilizat\* la clasificare, similar cu cel din fig. 5.4. Codul sursă este scris în Rust pentru a beneficia de performanţa şi siguranţa limbajului, iar în scopuri demonstrative este o inclusă o versiune abstractizată a algoritmului de procesare DEX (fig. 5.3).

```

1 {
2   "signature": "Landroidx/core/content/FileProvider$Api21Impl;getExternalMediaDirs(Landroid/content/Context;)[Ljava/io/File;",
3   "ins": [
4     { "op": "InvokeVirtual", "id": 548},
5     { "op": "MoveResultObject"},
6     { "op": "ReturnObject"}
7   ],
8 },
9 {
10  "signature": "Landroidx/core/content/FileProvider$SimplePathStrategy;<init>(Ljava/lang/String;)V",
11  "ins": [
12    { "op": "InvokeDirect", "id": 32188},
13    { "op": "NewInstance"},
14    { "op": "InvokeDirect", "id": 32999},
15    { "op": "InputObject"},
16    { "op": "InputObject"},
17    { "op": "ReturnVoid"}
18  ],
19 },
20 {
21  "signature": "Landroidx/core/content/FileProvider$SimplePathStrategy;addRoot(Ljava/lang/String;Ljava/io/File;)V",
22  "ins": [
23    { "op": "InvokeStatic", "id": 1811},
24    { "op": "MoveResult"},
25    { "op": "IfNez"},
26    { "op": "InvokeVirtual", "id": 31833},
27    { "op": "MoveResultObject"},
28    { "op": "IgetObject"},
29    { "op": "InvokeVirtual", "id": 33088},
30    { "op": "ReturnVoid"},
31    { "op": "MoveException"},
32    { "op": "NewInstance"},
33    { "op": "NewInstance"},
34    { "op": "InvokeDirect", "id": 32299},
35    { "op": "ConstString"},
36    { "op": "InvokeVirtual", "id": 32311},
37    { "op": "InvokeVirtual", "id": 32310},
38    { "op": "InvokeVirtual", "id": 32326},
39    { "op": "MoveResultObject"},
40    { "op": "InvokeDirect", "id": 32099},
41    { "op": "Throw"},
42    { "op": "NewInstance"},
43    { "op": "ConstString"},
44    { "op": "InvokeDirect", "id": 32098},
45    { "op": "Throw"}
46  ],
47 }

```

Figura 5.4: Exemplu de secvenţă de metode extrasă din aplicaţia F-Droid<sup>6</sup> (JSON)

### 5.3.3 Preprocesare

Dat fiind faptul că biblioteca de extragere a datelor returnează o instanţă a structurii *Apk*, şi modelele machine learning nu înţeleg structuri de date

complexe, este necesară transformarea *Apk*-ului într-un input care poate fi folosit de model. Această parte nu este implementată în bibliotecă, ci este un modul separat din aplicația web.

Modelul nu poate accepta set-ul de permisiuni ca un array de string-uri. Acesta este convertit într-un vector binar de dimensiunea 50, unde fiecare poziție înseamnă prezența sau absența unei permisiuni în *Android-Manifest.xml*:

- 1 - Permisiunea este declarată
- 0 - Permisiunea nu este declarată

Aplicațiile Android pot declara sute de permisiuni, dar în practică unele permisiuni sunt foarte rar întâlnite și nu sunt relevante. Pentru eliminarea permisiunilor redundante a fost folosit un clasificator Random Forest<sup>413</sup>. Setul total de 368 de permisiuni întâlnite în dataset, a fost redus, folosind codul Python din fig. 5.5, la 50 de permisiuni cele mai relevante (fig. 5.6), top 5 fiind:

1. `SEND_SMS` - Permite trimiterea mesajelor SMS.
2. `RECEIVE_SMS` - Permite citirea mesajelor SMS.
3. `MOUNT_UNMOUNT_FILESYSTEMS` - Permite montarea și demontarea sistemului de fișiere.
4. `READ_PHONE_STATE` - Permite accesul (read-only) la starea telefonului, inclusiv la informațiile actuale ale rețelei celulare, la starea oricăror apeluri în curs și la o listă a oricăror conturi de telefon înregistrate pe dispozitiv.
5. `ACCESS_WIFI_STATE` - Permite accesarea informațiilor despre rețelele Wi-Fi.

```

1 # Import biblioteci necesare
2 import os
3 import json
4 import keras
5 import numpy as np
6 import matplotlib.pyplot as plt
7 from tqdm import tqdm
8 from sklearn.ensemble import RandomForestClassifier
9 from sklearn.preprocessing import MultiLabelBinarizer
10
11 # Încărcare date în memorie
12 categories = sorted(['Adware', 'Banking', 'Benign', 'Riskware', 'SMS'])
13 X, y = [], []
14 for i, category in enumerate(categories):
15     files = list(os.listdir(f'dataset/{category}'))
16     for file in tqdm(files, desc=category):
17         with open(f'dataset/{category}/{file}', 'rb') as f:
18             data = json.load(f)
19             if (manifest := data['man']) is not None:
20                 X.append(manifest['prm'])
21                 y.append(i)
22
23 # Preprocesare
24 mlb = MultiLabelBinarizer()
25 X = mlb.fit_transform(X)
26 y = keras.utils.to_categorical(y)
27
28 # Instanțiere Random Forest Classifier
29 rf = RandomForestClassifier(n_estimators=10, random_state=42)
30 rf.fit(X, y)
31
32 # Plotare top 50 permisiuni
33 plt.figure(figsize=(10, 10))
34 importances = rf.feature_importances_
35 indices = np.argsort(importances)[::-1]
36 plt.barh(range(50), importances[indices[:50]])
37 plt.yticks(range(50), mlb.classes_[indices[:50]])
38 plt.show()

```

Figura 5.5: Selectarea celor mai importante 50 de permisiuni

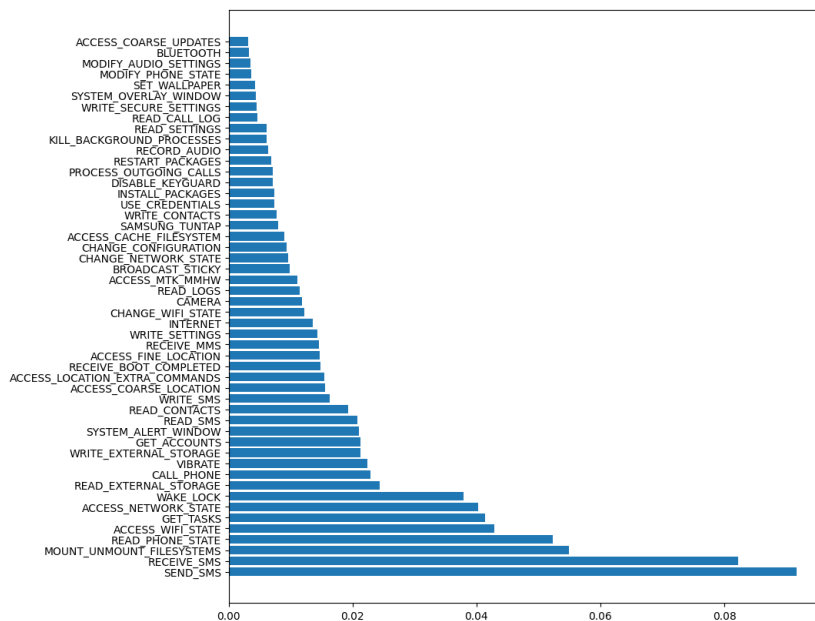


Figura 5.6: Top 50 cele mai importante permisiuni

La fel ca și cu permisiunile, metodele nu pot fi transmise direct. Vectorul de metode este convertit în doi vectori separați:

- **Opcode Vector** - Secvența de opcode-uri concatenată a tuturor metodelor.
- **Indices Vector** - Vector de indici [start, end] care delimitează metodele în **Opcode Vector**.

Pe lângă aceasta mai e de menționat că vectorii vor fi truncați astfel încât ultimul index de start de metodă să fie sub valoarea 512000 (fig. 5.7) pentru a preveni supraîncărcarea memoriei.

Astfel vom avea un vector de dimensiune fixă 50 cu valori 0, 1 pentru permisiuni și doi vectori de dimensiune variabilă pentru bytecode-ul aplicației.

### 5.3.4 Testare

Pentru asigurarea corectitudinii parsării APK-urilor, a fost elaborată o suită de teste unitare folosind biblioteca standard din Rust. În total au fost scrise 10 teste unitare:

1. `dex::instruction::tests::test_empty_bytecode`
2. `dex::instruction::tests::test_bad_opcode`

```

1 let mut opcodes = Vec::new();
2 let mut indices = Vec::new();
3 let mut index = 0;
4 for method in &apk.methods {
5     let tmp = index + method.insns.len();
6     opcodes.extend(method.insns.iter().map(|inst| inst.opcode as u8));
7     indices.push(index);
8     indices.push(tmp - 1);
9     index = tmp;
10    if index >= 512_000 {
11        break;
12    }
13 }

```

Figura 5.7: Truncarea secvenței bytecode pentru un APK

3. dex::instruction::tests::test\_invoke\_args
4. dex::instruction::tests::test\_too\_short
5. dex::instruction::tests::test\_valid\_opcode
6. dex::tests::test\_hello\_world
7. dex::tests::test\_call\_graph
8. manifest::tests::test\_parse\_a
9. manifest::tests::test\_parse\_b
10. manifest::tests::test\_parse\_c

Testele 1-5 asigură corectitudinea parsării instrucțiunilor individuale din cadrul unei secvențe de octeți.

Testele 6-7 verifică ordinea și integritatea metodelor extrase din fișiere DEX. Fișierele DEX au fost generate manual prin compilarea de cod Java și conversia rezultatului binar în format DEX folosind instrumentul de linie de comandă *d8*<sup>19</sup>.

Testele 8-10 asigură corectitudinea parsării componentelor și permisiunilor din fișiere *AndroidManifest.xml*. Fișierele au fost obținute prin build-area manuală a unor proiecte Android simple pentru a obține fișiere .apk, care au fost dezarhivate pentru a extrage *AndroidManifest.xml* în formatul lor binar.

## 5.4 Clasificare

### 5.4.1 Dataset

În cadrul acestei lucrări, a fost folosit datasetul CICMalDroid 2020<sup>21,22</sup> pentru antrenarea și testarea modelului. Setul de date conține 17341 de aplicații Android colectate în perioada Decembrie 2017 - Decembrie 2018 din cinci categorii:

- **Adware** - Afișează material publicitar nesolicitat.
- **Banking** - Obține acces la conturile bancare online ale utilizatorului, imitând aplicațiile bancare originale sau interfața web bancară. Majoritatea sunt concepute pentru a se infiltra în dispozitive și fura detalii sensibile, de exemplu, loginul și parola contului bancar și pentru a trimite informațiile furate către un server de comandă și control (C&C).
- **SMS** - Exploatează serviciul SMS, pentru a trimite sau intercepta mesaje cu scopul răspândirii virusului și exfiltrării datelor.
- **Riskware** - Pot cauza daune dacă utilizatorii rău intenționați le exploatează. Se poate transforma în orice altă formă de malware, cum ar fi Adware sau Ransomware, care extinde funcționalitățile instalând aplicații nou infectate.
- **Benign** - Aplicații care nu sunt rău intenționate.

După preprocesare, doar 16457 de aplicații au fost decompilate cu succes, rezultând numerele finale din fig. 5.8.

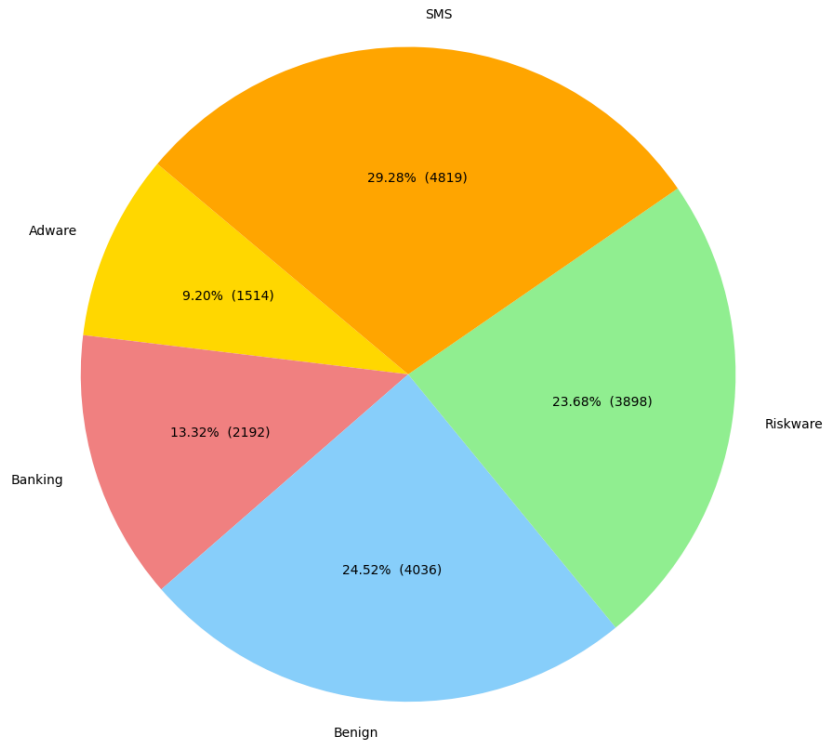


Figura 5.8: Distribuția finală a categoriilor în setul de date

Pentru antrenare a fost folosit un subset de 80% din date (din care 20% pentru validare) și 20% pentru testare.

### 5.4.2 Model

La implementării din lucrare stă arhitectură ierarhică de blocuri Perceive-rIO<sup>11</sup> a modelului Malceiver<sup>23</sup> care folosește 3 feature-uri la nivel de cod:

1.  $C$  - Caracteristicile locale la nivel de relat, ii între opcodes învecinați
2.  $M$  - Caracteristicile la nivel de metodă
3.  $g$  - Caracteristicile la nivel global

### 5.4.3 Nivel local

Secvența de opcode-uri este trecută printr-un strat Embedding pentru a învăța reprezentările semantice a fiecărui opcode. Acestor reprezentări le este aplicată o convoluție pentru a captura relațiile dintre opcode-uri și a comprima secvența<sup>8</sup>.

Astfel se obține o matrice de filtre convoluționale  $C \in \mathbb{R}^{n \times c}$ , unde  $n$  corespunde cu lungimea secvenței iar  $c$  corespunde cu numărul de filtre convoluționale. Aceasta este introdusă în primul bloc PerceiverIO care va returna matricea latentă pentru următorul bloc.

#### 5.4.4 Nivel de metodă

Se calculează suma cumulată a matricii  $C$ , și se obține matricea  $\hat{C} \in \mathbb{R}^{n \times c}$ . Apoi se calculează o sinteză a fiecărei metode folosind vectorul de indici  $\mathcal{F}$  și matricea  $\hat{C}$ :

$$M = \left( \frac{\hat{C}_{e_f} - \hat{C}_{s_f}}{e_f - s_f} \right), (s_f, e_f) \in \mathcal{F}$$

Prin aceasta se obține matricea  $M \in \mathbb{R}^{F \times c}$ ,  $F$  fiind numărul de metode din APK. Matricea conține sinteze a tuturor metodelor, fiecare fiind normalizată folosind lungimea sa. Se introduce matricea în al doilea bloc PerceiverIO care va returna matricea latentă pentru următorul bloc.

#### 5.4.5 Nivel global

Se aplică Global Max-Pooling peste matricea  $C$ , și se obține un vector de caracteristici  $g \in \mathbb{R}^{1 \times n}$  care reprezintă o sinteză globală bytecode-ului. Se introduce în al treilea bloc PerceiverIO care va returna matricea latentă pentru concatenare cu output-ul de la Multi Layer Perceptron (MLP)<sup>9</sup> al permisiunilor.



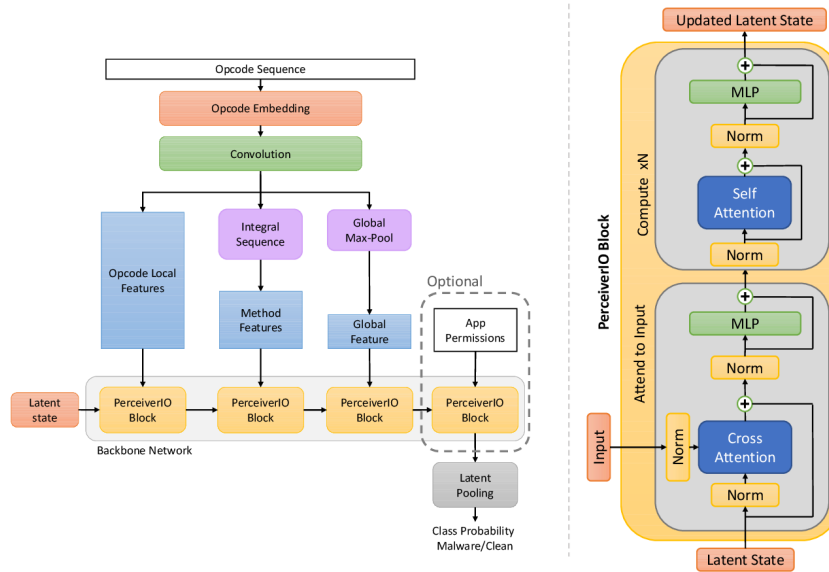


Figura 5.9: Arhitectura Rețelei Malceiver<sup>23</sup> (Figura 1) *arXiv:2204.05994*, 2022. doi:10.48550/arXiv.2204.05994.

### 5.4.6 Permisuni

Una din deviațiile de la implementarea originală a modelului Malceiver din fig. 5.9 constă în abordarea diferită a permisiunilor. În loc să fie folosit un strat Embedding și încă un bloc PerceiverIO, se folosește un simplu MLP, rezultatul fiind concatenat cu matricea latentă finală la care s-a aplicat un Global Average Pooling. În urma experimentării sa depistat că această concatenare a dat rezultate mai bune, iar numărul de parametri a fost redus.

### 5.4.7 Output

Vectorul concatenat este unit cu un strat alcătuit din 5 neuroni corespunzători fiecărei categorii. Funcția de activare folosită este *sigmoid* pentru fiecare neuron, iar funcția de loss este *Binary Crossentropy*. Aceasta a fost o modificare minoră față de implementarea originală pentru a adapta modelul la clasificare multi-label în loc de binară. Prin unirea tuturor blocurilor este obținut modelul final din fig. 5.10 antrenat pe datasetul CICMalDroid 2020<sup>21,22</sup>.

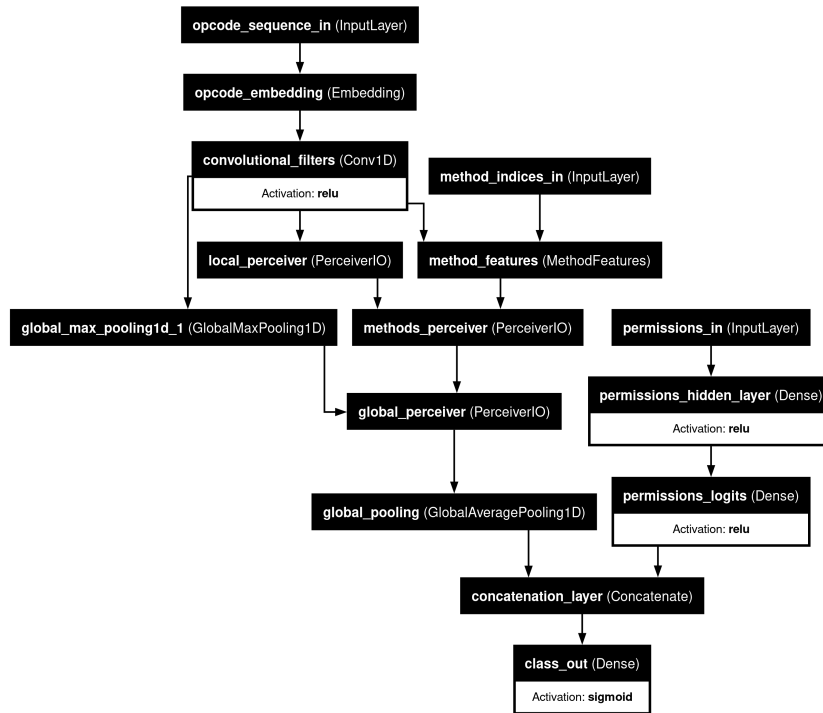


Figura 5.10: Modelul final

#### 5.4.8 Antrenare

Antrenarea a fost realizată folosind biblioteca Keras<sup>5</sup> pentru Python. Keras simplifică antrenarea modelului prin expunerea unor funcționalități precum callback-uri. Astfel este posibilă o antrenare fără cod redundant și cu o structură clară ca în fig. 5.11.

Pentru a asigura obținerea celui mai performant model, a fost folosit callback-ul *ModelCheckpoint*. *ModelCheckpoint* salvează modelul/ponderile într-un fișier dacă metrica prestabilită este îmbunătățită. În cazul modelului elaborat, metrica prestabilită a fost *val\_accuracy* (acuratețea pe setul de validare).

Prevenirea overfitting-ului este realizată de callback-ul *EarlyStopping*. *EarlyStopping* oprește antrenarea dacă metrica prestabilită (*val\_accuracy*) nu se îmbunătățește într-un număr de epoci consecutiv, 5 în cazul modelului elaborat.

```

1 import keras
2
3 filepath = datetime.now().strftime('%Y%m%d%H%M%S')
4 mcc = keras.callbacks.ModelCheckpoint(
5     filepath=f'saved/{filepath}.weights.h5', monitor='val_accuracy',
6     mode='max', save_best_only=True, save_weights_only=True
7 )
8 es = keras.callbacks.EarlyStopping(monitor='val_accuracy', mode='max', patience=5)
9 history = model.fit([X_train, I_train, P_train], y_train, validation_split=.2, epochs=n_epochs, callbacks=[mcc, es])

```

Figura 5.11: Snippet cu antrenarea modelului

## 6. Rezultate experimentale

### 6.1 Descrierea procesului de testare

### 6.2 Rezultate experimentale

## **7. Concluzii, contribuții și direcții de continuare a dezvoltării**

### **7.1 Concluzii**

### **7.2 Contribuții**

### **7.3 Direcții viitoare**

# Bibliografie

- [1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL <https://www.tensorflow.org/>. Software available from tensorflow.org.
- [2] Hybrid Analysis. Free automated malware analysis service. <https://www.hybrid-analysis.com/>, 2024. Accesat: 25 mai 2024.
- [3] Ayrx. axmldecoder. <https://github.com/Ayrx/axmldecoder>, 2024.
- [4] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001. doi: 10.1023/A:1010933404324.
- [5] François Chollet et al. Keras. <https://keras.io>, 2015.
- [6] F-Droid Contributors. F-droid. <https://f-droid.org/>, 2024. Accesat: 26 mai 2024.
- [7] The Rust Project Developers. Rayon: A data parallelism library for rust. <https://github.com/rayon-rs/rayon>, 2024. URL <https://github.com/rayon-rs/rayon>. Version 1.10.0.
- [8] Ali Hassani, Steven Walton, Nikhil Shah, Abulikemu Abuduweili, Jia-chen Li, and Humphrey Shi. Escaping the big data paradigm with compact transformers, 2022.
- [9] Simon Haykin. *Neural networks: a comprehensive foundation*. Prentice Hall PTR, 1994.
- [10] MongoDB Inc. MongoDB: The developer data platform. <https://www.mongodb.com/>, 2024. Accesat: 25 mai 2024.
- [11] Andrew Jaegle, Sebastian Borgeaud, Jean-Baptiste Alayrac, Carl Doersch, Catalin Ionescu, David Ding, Skanda Koppula, Daniel Zoran,

- Andrew Brock, Evan Shelhamer, Olivier Hénaff, Matthew M. Botvinick, Andrew Zisserman, Oriol Vinyals, and João Carreira. Perceiver io: A general architecture for structured inputs & outputs, 2022.
- [12] Jotti. Jotti’s malware scan. <https://virusscan.jotti.org/>, 2024. Accesat: 25 mai 2024.
  - [13] Kartik Khariwal, Rishabh Gupta, Jatin Singh, and Anshul Arora. R-mfdroid: Android malware detection using ranked manifest file components. *International Journal of Innovative Technology and Exploring Engineering*, 10:55–64, 05 2021. doi: 10.35940/ijitee.G8951.0510721.
  - [14] Google LLC. Application fundamentals. <https://developer.android.com/guide/components/fundamentals>, 2024. Accesat: 19 mai 2024.
  - [15] Google LLC. Android architecture. <https://source.android.com/docs/core/architecture>, 2024. Accesat: 19 mai 2024.
  - [16] Google LLC. Permissions overview. <https://developer.android.com/guide/topics/permissions/overview>, 2024. Accesat: 18 mai 2024.
  - [17] Google LLC. Android runtime and dalvik. <https://source.android.com/docs/core/runtime>, 2024. Accesat: 25 mai 2024.
  - [18] Google LLC. Android open source project. <https://source.android.com>, 2024. Accesat: 19 mai 2024.
  - [19] Google LLC. d8. <https://developer.android.com/tools/d8>, 2024. URL <https://developer.android.com/tools/d8>. Accesat: 26 mai 2024.
  - [20] Google LLC. Dalvik executable format. <https://source.android.com/docs/core/runtime/dex-format>, 2024. Accesat: 26 mai 2024.
  - [21] Samaneh Mahdavifar, Andi Fitriah Abdul Kadir, Rasool Fatemi, Dima Alhadidi, and Ali A. Ghorbani. Dynamic android malware category classification using semi-supervised deep learning. In *2020 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCCom/CyberSciTech)*, pages 515–522, 2020. doi: 10.1109/DASC-PiCom-CBDCCom-CyberSciTech49142.2020.00094.

- [22] Samaneh MahdaviFar, Dima Alhadidi, and Ali Ghorbani. Effective and efficient hybrid android malware classification using pseudo-label stacked auto-encoder. *Journal of Network and Systems Management*, 30, 01 2022. doi: 10.1007/s10922-021-09634-4.
- [23] Niall McLaughlin. Malceiver: Perceiver with hierarchical and multi-modal features for android malware detection, 2022.
- [24] Inc OPSWAT. Metadefender cloud. <https://metadefender.opswat.com/>, 2024. Accesat: 25 mai 2024.
- [25] Hispasec Sistemas. Virustotal. <https://www.virustotal.com/>, 2024. Accesat: 25 mai 2024.
- [26] The Rust Team. The rust programming language. <https://www.rust-lang.org/>, 2024. Accesat: 18 mai 2024.
- [27] tokio rs. axum. <https://github.com/tokio-rs/axum/>, 2024. Accesat: 18 mai 2024.
- [28] Jordan Walke. React. <https://react.dev/>, 2024. Accesat: 18 mai 2024.
- [29] Wikipedia. apk (file format). [https://en.wikipedia.org/wiki/Apk\\_\(file\\_format\)](https://en.wikipedia.org/wiki/Apk_(file_format)), 2024. Accesat: 18 mai 2024.
- [30] Wikipedia. Depth-first search. [https://en.wikipedia.org/wiki/Depth-first\\_search](https://en.wikipedia.org/wiki/Depth-first_search), 2024. Accesat: 26 mai 2024.