România
Ministerul Apărării Naționale
Academia Tehnică Militară "*Ferdinand I*"

Facultatea de Sisteme Informatice și Securitate Cibernetică
Ingineria și Securitatea Sistemelor Informatice Militare

Platformă de Analiză Automată a Aplicațiilor Malițioase prin Utilizarea unor Algoritmi de Inteligență Artificială

Coordonator Științific
Lect. Univ. Dr. Ing. Alin PUNCIOIU

Absolvent
Sd. Sg. Maj. George-Andrei IOSIF

București
2021

Aș dori să adresez mulțumiri domnului Alin Puncioiu, coordonatorul acestei lucrări, pentru îndrumarea și libertatea oferite. Proiectul a ajuns în forma actuală și datorită domnilor Radu Velea și Ion Bădoi, care au făcut posibilă integrarea soluției software dezvoltate în cadrul unui proiect de cercetare al Academiei Tehnice Militare "*Ferdinand I*" București.

În plus, le sunt profund recunoscător părinților mei, Cristianei și lui Iulian pentru că m-au sprijinit moral pe tot parcursul scrierii acestei lucrări.

Abstract

The manual analysis of malicious programs is getting even more demanding because of their increase in volume and complexity. Therefore, automated solutions based on a series of static and dynamic heuristics have been introduced to sort through the dense stream of files sent for analysis. The latest approaches propose the usage of artificial intelligence algorithms, which automatically infer models and patterns that can not be easily observed (and later implemented in heuristics) by the analysts.

The present paper has the purpose of showing scenarios in which artificial intelligence brings visible benefits to the analysis of malicious programs. It starts from the presentation of introductory concepts, about malicious applications and artificial intelligence. Further, it details how a platform has been implemented to encompass these two areas, as well as the aspects that have been considered in the development process: pipelining, the division into modules, distributed computing, and configurability, to meet the needs of the end users.

Rezumat

Analiza manuală a programelor malițioase devine din ce în ce mai solicitantă din cauza creșterii în volum și în complexitate a acestora. Prin urmare, s-au introdus soluții automate care se bazează pe o serie de euristici statice și dinamice pentru a tria fluxul dens de fișiere trimise spre a fi analizate. Ultimele abordări propun folosirea de algoritmi de inteligență artificială, ce deduc automat modele și șabloane ce nu pot fi observate cu ușurință (și ulterior implementate în euristici) de către analiști.

Această lucrare are scopul de a prezenta scenarii în care folosirea inteligenței artificiale aduce beneficii vizibile analizei de programe malițioase. Ea pleacă de la prezentarea unor noțiuni introductive, despre aplicații malițioase și inteligență artificială. Ulterior, detaliază modul în care o platforma a fost implementată pentru a îngloba aceste două domenii, cât și aspectele de care s-a ținut cont în procesul de dezvoltare: *pipelining*, împărțirea pe module, calcul distribuit și configurativitate, pentru a satisface nevoile utilizatorilor finali.

Tabelă de Conținut

Listă de Imagini

Listă de Abrevieri

Toate abrevierile de mai jos provin din limba engleză.

API ..... *Application Programming Interface*

ICA ..... *Independent Component Analysis*

IP ..... *Internet Protocol*

ISA ..... *Instruction Set Arhitecture*

NMF ..... *Non-negative Matrix Factorization*

OLE ..... *Object Linking and Embedding*

PCA ..... *Principal Component Analysis*

PE ..... *Portable Executable*

RAM ..... *Random Access Memory*

RPC ..... *Remote Procedure Call*

TCP ..... *Transmission Control Protocol*

TTP ..... *Techniques, Tactics and Procedures*

VBA ..... *Visual Basic for Applications*

Capitolul 1: Introducere

1.1 Context

Încă din secolul al XIX-lea, când Charles Babbage a descris primul mecanism capabil să soluționeze automat diferite tipuri de probleme, calculatoarele au evoluat constant. De la acele roți zimțate, s-a trecut ulterior la tuburi vidate și la tranzistori, ultimii fiind grupați inițial în circuite integrate limitate din punct de vedere al numărului lor. În prezent, calculatoarele dețin cipuri electronice fabricate cu ajutorul integrării la scară largă (engl. "*large scale integration*"), având mii de tranzistoare pe centimetru pătrat de silicon. Ele ne ușurează viețile prin avantajele pe care le aduc: automatizarea sarcinilor repetitive sau dificile, stocarea unui volum mare de date și interconectarea cu oameni de pe tot globul pământesc.

Deși această descriere ar putea părea una utopică, trebuie să luăm în considerare și dezavantajele pe care tehnologia le implică, precum **amenințările spațiului cibernetic**. Cum în fiecare societate, oricât de primordială sau de dezvoltată ar fi ea, au existat de fiecare dată oameni care nu s-au conformat regulilor impuse la nivelul acesteia, același lucru s-a întâmplat și în spațiul cibernetic. A apărut treptat, o dată cu facilitarea accesului la calculatoare, fenomenul de **crimă cibernetică**, în care persoane rău intenționate folosesc tehnologia pentru a dăuna siguranței cuiva sau pentru a obține beneficii materiale. Tehnicile pe care acești criminali cibernetici le utilizează variază în prezent de la compromiterea de pagini web pentru acces neautorizat la datele confidențiale ale utilizatorilor și până la folosirea de programe cu caracter malițios, pentru compromiterea completă a serverelor sau a dispozitivelor folosite de oamenii obișnuiți.

Cum această creștere a devenit prea accentuată, depășind limitele confortabile în care o persoană specializată și o soluție de securitate bazată pe semnături puteau proteja o rețea informațională, respectiv dispozitive folosite de utilizatori finali, a apărut necesitatea de a avea **soluții software complexe și automate**, bazate atât pe euristici statice, cât și pe unele dinamice. Acestea au însă dezavantajul rigidității, întrucât ele nu mai evoluează decât prin intermediul unor actualizări care uneori pot fi prea târzii.

Astfel, **domeniul inteligenței artificiale**, în special cel de învățare automată pe care îl cuprinde, a fost integrat în aceste soluții pentru a oferi maleabilitatea necesară adaptării la mediului dinamic în care ele funcționează. Modul de interpretare al formatului de fișier a rămas aproximativ același, însă diferențe apar la limitele impuse de euristicile clasice, care au fost înlocuite de unele deduse în mod automat de algoritmi inteligenți.

1.2 Stadiul Actual al Tehnologiei

Autorii "*Survey of Machine Learning Techniques for Malware Analysis*" [3] au realizat în **anul 2019** un **studiu asupra modurilor** în care tehnicile de învățare automată au fost folosite pentru analiza programelor malițioase. Sistemul de operare studiat a fost Windows întrucât acesta este cel mai folosit la nivel mondial pe dispozitivele de tip *desktop*[1]. Studiul se remarcă prin parcurgerea unui număr mare de lucrări publicate și deducerea unor aspecte precum:

- **Obiectivele**, care relevă tipul de ieșire produs de către soluțiile dezvoltate de către cercetătorii studiilor, se încadrează în detecția programelor malițioase și a familiilor de care aparțin, cât și în analiza similarității unui exemplar, prin compararea lui cu altele, aflate într-un set de date.
- Soluțiile folosesc **atribute** extrase atât static, cât și dinamic. Unele soluții le folosesc pe ambele, într-o abordare numită hibridă.
- Partea de învățare automată este efectuată prin intermediul tuturor **tipurilor de algoritmi**: supervizați, nesupervizați și semi-supervizați.

1.3 Motivația Lucrării

Un lucru interesant pe care l-am putut observa din verificarea lucrărilor pe care studiul de mai sus le menționează este că **majoritatea nu oferă o soluție** (produs software de sine stătător sau bibliotecă de cod) care să poată fi folosită la scară largă de către posibili beneficiari ai proiectelor lor, de la utilizatorii simpli de dispozitive mobile și de calculatoare personale și până la organizații posesoare de servere și de stații de lucru.

Fiecare studiu își alege un mod de a rezolva problema analizei de programe malițioase, printr-o pereche formată din mulțimea de atribute extrase și din algoritmii de învățare automată ce au fost aplicați. Însă nu reușesc să aducă aproape de utilizatorii finali metodele de analiză pe care le propun, prin oferirea unui cadru ce lasă loc dezvoltărilor ulterioare și integrării de alte abordări.

Pe lângă aceste probleme ce țin de interacțiunea dintre lumea academică și mediul în care organizațiile funcționează, contextul actual de securitate cibernetică impune o atenție ridicată anumitor aspecte. Trebuie avut în considerare **factorul uman** reprezentat de membrii organizației pentru că el este implicat

[1] https://gs.statcounter.com/os-market-share/desktop/worldwide

în majoritatea atacurilor cibernetice[2]. Întrucât multe compromiteri pleacă de la descărcări și rulări de fișiere malițioase, putem deduce importanța pe care o are o protecție asupra **formatului de fișiere executabile** folosit pentru sistemele de operare Windows, cât și a aceluia pentru **platformele de tip Office**[3], ce sunt folosite des de membrii oricărui tip de organizații.

Din punct de vedere al analiștilor care fac parte din departamentele de securitate cibernetică ale organizațiilor, putem sublinia faptul că **analiza manuală a unui fișier malițios modern**, pe care ei o efectuează, durează mult întrucât ele apelează la mecanisme de protejare împotriva analizei, care îngreunează procesul.

În plus, **contextul** este unul **foarte dinamic**. Criminalii cibernetici pot recurge la soluții automate de generare a mai multor variante ale aceluiași program malițios, comportament cu care pot eluda cu ușurință o soluție de securitate cu euristici simpliste.

Pe de altă parte, am putut identifica soluții comerciale ce folosesc tehnici de inteligență artificială, precum Kaspersky[4], avast[5] și McAfee[6]. Principalul dezavantaj îl reprezintă însă **povara financiară** a subscripției sau a cumpărării, ce nu poate fi suportată cu ușurință de o organizație mică, aflată la început de drum.

1.4 Obiectivele Lucrării

Plecând de la problemele evidențiate anterior, **ne propunem în această lucrare să demonstrăm că rezultatele analizei de programe malițioase sunt avantajate, prin perspectiva rezultatelor, de folosirea unor tehnici de inteligență artificială**, în special de învățare automată. Facem acest lucru prin **proiectarea unei platforme**, ce plecă de la un set de cerințe funcționale și nefuncționale esențiale. O prezentăm atât la nivel micro, din punct de vedere al modulelor ce o alcătuiesc, cât și holistic, prin efectuarea unor teste și prin evaluări ale rezultatelor.

Astfel, scopul intrinsec al platformei este de a reprezenta o soluție software cu sursă deschisă (engl. "*open-source*"), ușor adaptabilă contextului dinamic al securității cibernetice, ce poate fi folosită pentru apărarea detrimentului virtual al unei organizații. Ea este ușor de gestionat într-o infrastructură modernă, integrabilă cu alte sisteme datorită existenței unor interfețe programabile și permite dezvoltarea ulterioară, prin oferirea unor interfețe pentru modulele componente.

Concret, ea automatizează modul de lucru specific analizei datelor pentru stabilirea unor euristici cu ajutorul cărora să poată fi scanate fișiere. În acest

[2]. *Data Breach Investigations Report.* `https://enterprise.verizon.com/resources/reports/2021-data-breach-investigations-report.pdf`. 2021

[3]. *Data Breach Investigations Report.* `https://enterprise.verizon.com/resources/reports/2020/2020-data-breach-investigations-report.pdf`. 2020

[4]https://www.kaspersky.com/enterprise-security/wiki-section/products/machine-learning-in-cybersecurity

[5]https://www.mcafee.com/enterprise/en-us/solutions/machine-learning.html

[6]https://www.avast.com/technology/ai-and-machine-learning

moment, suportă numai sistemul de operare Windows și formatele executabile, *Portable Executable* (abreviat PE), și cele Office, *Object Linking and Embedding* (abreviat OLE). Platforma ajută analiștii pentru luarea unor decizii rapide cu privire la exemplare malițioase întâmpinate și pentru crearea unor premise (maliție, familii de programe malițioase și fișiere similare) de la care ei să înceapă o analiză profundă. Alți beneficiari din cadrul organizației sunt ceilalți angajați, care își pot scana cu ușurință, prin intermediul unei interfețe facile, fișierele pe care le folosesc în munca lor de zi cu zi.

1.5 Structura pe Capitole

Lucrarea este împărțită în **șase capitole**. Primul, cel de față, prezintă contextul general al securității cibernetice, problemele pe care le considerăm esențiale motivării eforturilor noastre și obiectivele lucrării. Capitolul al doilea prezintă noțiuni teoretice, plecând de la cele generice, despre știința calculatoarelor și ajungând până la formatele de fișiere procesate automat, și anume PE și OLE. Urmează noțiuni despre analiza de programe malițioase, inteligență artificială și, în ultimă instanță, despre alte tematici ce au fost folosite în dezvoltarea soluției software.

Începând cu al treilea capitol, prezentăm modul în care platforma de analiză automată a fost proiectată. Cerințele funcționale și nefuncționale vor reprezenta fundația modulelor detaliate în continuarea capitolului, ce se încheie cu modalitatea de testare. În cel de-al patrulea capitol, expunem metodologia de evaluare a platformei dezvoltate, cât și rezultatele acesteia.

În penultimul capitol, concluzionăm cele expuse în toate capitolele anterioare și trasăm idei cu privire la dezvoltarea ulterioară a platformei. Ultimul, cel de-al șaselea, este cel de bibliografie.

Capitolul 2:
Noțiuni Teoretice

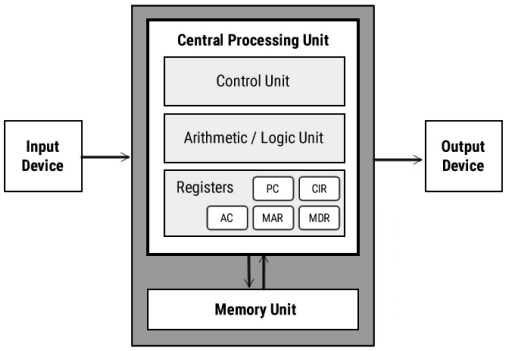
În acest capitol, vom prezenta **noțiunile teoretice necesare înțelegerii funcționării platformei propuse**. Concret, vom vorbi despre știința calculatoarelor pentru a avea contextul general al funcționării acestor dispozitive. Vom continua cu descrierea unor concepte de programe malițioase și de inteligență artificială, cu accent pe învățarea automată, urmând în finele capitolului să prezentăm succint noțiunile rămase, ce țin de funcționalitatea, arhitectura și implementarea platformei.

2.1 Noțiuni Introductive

Știința calculatoarelor reprezintă **studiul informației, a protocoalelor și a algoritmilor** folosiți pentru **automate reale și idealizate**. Ea cuprinde o serie de domenii, ce pot fi împărțite în **teoretice**, precum teoria computației, ce studiază modele și o serie de probleme ce pot fi rezolvate cu ajutorul lor, și **practice**, precum grafica ce analizează modele digitale de sintetizare și de manipulare a contextului grafic.

2.1.1 Organizarea Internă a Calculatorului

Modelul matematic prezentat de Alan Turing a reprezentat punctul de plecare al dezvoltării unor calculatoare digitale. În ciuda acestui fapt, în zilele noastre este folosită **arhitectura propusă de John von Neumann**, cunoscută sub numele de von Neumann sau Princeton. Aceasta cuprinde o serie de componente care face posibilă funcționarea oricărui calculator cu programe salvate, în care preluările de instrucțiuni și operațiile cu date nu pot apărea în același timp din cauza magistralei comune.



Imagine 2.1: Arhitectura von Neumann a unui calculator[1]

Calculatorul poate fi privit ca având o **funcționare în trei etape**: intrarea datelor, procesarea și ieșirea datelor procesate. Astfel, prin intermediul unor dispozitive speciale, se preiau anumite stări sau acțiuni din mediul extern, în care calculatorul se află. Acestea sunt transportate prin intermediul unei magistrale către unitatea centrală, unde ele sunt procesate. Odată ce procesarea s-a finalizat, rezultatele sunt fie salvate în memoria volatilă ("*random-access memory*" și abreviat RAM), prin intermediul unității de gestiune a ei și pentru a fi utilizate ulterior pentru o altă procesare, fie întoarse în mediul exterior printr-o altă gamă de dispozitive speciale, de ieșire.

În arhitectura prezentată, **unitatea centrală de procesare** joacă cel mai important rol. Numită și procesor, este un circuit electronic construit în mare parte din tranzistori pe bază de silicon și capabil de a executa instrucțiunile ce formează un program. Multe procesoare sunt de natură sincronă, fapt ce implică un semnal de ceas pentru a impune un ritm operațiilor secvențiale executate. Semnalul este produs de un circuit oscilator extern, ce generează un număr constant de pulsuri electrice pe secundă.

Ciclul de lucru al procesorului presupune rularea în continuu a unei bucle formate din aducerea din memorie și decodificarea instrucțiunilor, aducerea din memorie a datelor necesare, execuția instrucțiunii și actualizarea opțională a memoriei. Cum de fiecare astfel de etapă se ocupă o unitate specializată a procesorului, o optimizare de tip linie de asamblare presupune **execuția paralelă a etapelor** astfel încât fiecare unitate să fie ocupată, în fiecare moment, cu procesarea unei instrucțiuni. În cel mai favorabil scenariu, acestă optimizare susține o rată de o instrucțiune pe fiecare ciclu de semnal de ceas.

[1] https://www.computerscience.gcse.guru/theory/von-neumann-architecture

2.1.2 Arhitecturi de Procesoare

Abstractizarea modelului procesorului se realizează prin **arhitectura setului de instrucțiuni** ("*instruction set arhitecture*" și abreviat ISA), definit de obicei ca "*arhitectura procesorului*". Cum ISA oferă o interfață între hardware și software, sunt permise mai multe implementări ale ei, care pot varia în performanță, dimensiunea fizică și cost (timp, consum de energie electrică).

Cele mai uzuale arhitecturi pentru calculatoarele de uz general sunt ale companiilor Intel (în special IA-32 și IA-64) și Advanced Micro Devices. În același timp, Advanced RISC Machine sunt întâlnite pe dispozitive precum telefoane inteligente și tablete.

O astfel de arhitectură este definită, în general, de următoarele aspecte:

1. ***Clasa***: Definește uzul operanzilor folosiți, putând fi cu regiștrii de uz general, unde operanzii pot fi regiștrii sau locații de memorie, sau de tip încărcare-stocare, în care memoria poate fi accesată numai prin instrucțiuni speciale de lucru cu memoria.

2. ***Adresarea memoriei***: Definește modul de interogare al memoriei, fiind de obicei la nivel de octet. Unele arhitecturi necesită ca obiectele din memorie să fie aliniate, adică adresa octetului de la care începe un astfel de obiect să fie divizibilă cu numărul de octeți pe care lucrează procesorul.

3. ***Modelele de adresare***: Modelele de adresare sunt folosite la specificarea adreselor operanzilor. De obicei, aceștia pot fi regiștrii, valori constante sau deplasate, unde o valoare fixă, a zonei în care se află, este adăugată pentru a obține adresa de memorie reală.

4. ***Operanzii***: Sunt descriși **de tip și de dimensiune**, acestea putând varia de la un octet (un caracter ASCII sau un `char`) și până la 8 octeți (`long integer`). Dimensiunea specifică arhitecturii este considerată ca fiind un `word`.

5. ***Operațiunile***: Operațiunile sunt împărțite în categorii, cele mai uzuale fiind de transfer de date, logice, aritmetice, de control și de lucru cu numere reale.

6. ***Instrucțiunile de control ale execuției programului***: Obligatorii pentru toate arhitecturile, sunt cele de salturi condiționale sau necondiționale, apeluri de proceduri și întoarcere la funcția apelantă.

7. ***Codificarea ISA***: Poate fi realizată în două maniere: una în care lungimea instrucțiunilor este fixă și una în care aceasta este variabilă. Ultima abordare dispune de avantajul de a avea o relație de proporționalitate directă între dimensiunea instrucțiunii și raritatea ei (cu cât o instrucțiune este statistic folosită mai mult, cu cât lungimea de codificare este mai mică). Astfel, codificarea ISA definește modul în care instrucțiunile sunt reprezentate în memorie, ca și coduri de operație formate dintr-un grup de biți, ce specifică operațiunea ce se dorește a fi executată, și un altul, ce indică informații suplimentare necesare operației, precum operanzii acesteia.

2.1.2.1 Regiștrii

Zonele de memorie interne ale procesorului sunt numite **regiștrii**. Comparativ cu alte tipuri de memorii (cache, RAM, disc) aflate la dispoziția procesorului, ei au o viteză de manipulare foarte mare, de ordinul nanosecundelor. Acest avantaj vine însă la pachet cu dezavantaje, ei fiind puțini și de o lungime mică, dictată de ISA.

Cei mai uzuali regiștrii pot fi grupați în funcție de scopul lor, astfel:

- **Uz general**, care salvează operanzi și rezultate ale diverselor operații;
- **Starea programului** și **control** (limitat) al procesorului; și
- **Salvarea adresei instrucțiunii** ce urmează să fie executată.

2.1.2.2 Memorie Virtuală. Paginare. Segmentare

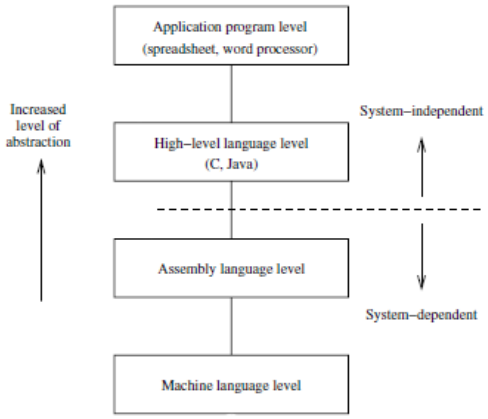
O problematică luată în considerare de arhitecții de procesoare reprezintă spațiul prea mare ocupat de mai multe procese (o astfel de entitate poate fi privită ca un program aflat în execuție la un anumit moment de timp), în condițiile în care acestea ocupă numai o mică parte din spațiul lor de adrese. Soluția găsită a fost **memoria virtuală**, ce împarte memoria RAM în blocuri, pe care le alocă diferitelor procese în funcție de nevoile lor. Aceste blocuri pot fi mutate pe disc în cazul în care sunt folosite rar, ele fiind aduse în memoria principală numai atunci când sunt accesate.

Mecanismul ce oferă suportul memoriei virtuale dispune de două tehnici: una ce lucrează cu blocuri de lungime fixă, numite **pagini**, și una cu blocuri de lungimi variabile, numite **segmente**. Decizia de a folosi o tehnică în favoarea alteia influențează viteza cu care procesorul funcționează, prin considerarea unor factori precum:

- Lungimea adresei, în `word`-uri, cu unul pentru pagini și două (identificatorul segmentului și deplasarea în segment) pentru segmente;
- Dificultatea înlocuirii unui bloc, fiind trivial pentru pagini datorită lungimii constante și dificil pentru segmente deoarece trebuie găsită o zonă continuă de memorie, neocupată; și
- Ineficiența folosirii memoriei, cu fragmentare internă pentru pagini și externă pentru segmente.

2.1.3 Limbaj de Asamblare

În funcție de abstractizarea dorită, se pot apela mai multe niveluri pentru a programa un procesor să execute o sarcină dată. Aceste niveluri formează o stivă, ce este ilustrată în figura de mai jos.

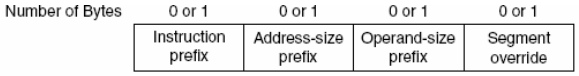


Imagine 2.2: Niveluri de interacțiune cu un sistem de calcul[2]

Cele mai înalte două niveluri limitează utilizatorul fie la lucrul cu o aplicație, fie la programarea într-un limbaj de nivel superior, pentru a-și scrie aplicații de unul singur. În ambele cazuri, nu este o necesară o cunoaștere detaliată a sistemului datorită independenței acestora două (de exemplu, de un procesor particular, utilizat în cadrul sistemului), excepție făcând cazul în care dezvoltarea implică noțiuni mai avansate, de exemplu de drivere. Pe de altă parte, limbajul de asamblare și cel mașină sunt de nivel inferior. Instrucțiunile lor permit realizarea unor sarcini de nivel mai jos comparativ cu cele menționate anterior.

Limbajul de asamblare reprezintă un mod de a afișa limbajul mașină, ce reprezintă numai o înșiruire de octeți ce pot fi înțeleși de către procesor. Pentru a realiza traducerea, limbajul de asamblare trebuie procesat de către un program numit **asamblor**.

Este format din instrucțiuni ce trebuie să respecte un anumit format, și anume `eticheta: mnemonic argumente`. Eticheta reprezintă o notație prin care alte instrucțiuni, precum cele de salt, pot face referire la instrucțiunea curentă. Mnemonicul este un nume rezervat pentru o clasă de instrucțiuni care au aceeași funcționalitate. Operanzii, având un număr variabil în funcție de codul operației, sunt folosiți pentru a executa instrucțiunea.



Imagine 2.3: Structura unei instrucțiuni pe arhitectura x86[3]

[2]Sivarama P Dandamudi. *Introduction to assembly language programming.* New York, NY: Springer, 2014

[3]http://www.c-jump.com/CIS77/CPU/x86/lecture.html

Deși programele dezvoltate în limbajul de asamblare au avantajul de a rula cu o viteză foarte mare pe procesoarele țintă, datorită folosirii optime a resurselor pe care acestea le oferă, ele sunt greu de dezvoltat, de întreținut și sunt dependente de ISA. Din aceste motive, s-a introdus un nivel nou care să îmbunătățească interfață cu resursele fizice ale calculatorului și să reducă timpul dezvoltării programelor: sistemul de operare

2.1.4 Sisteme de Operare

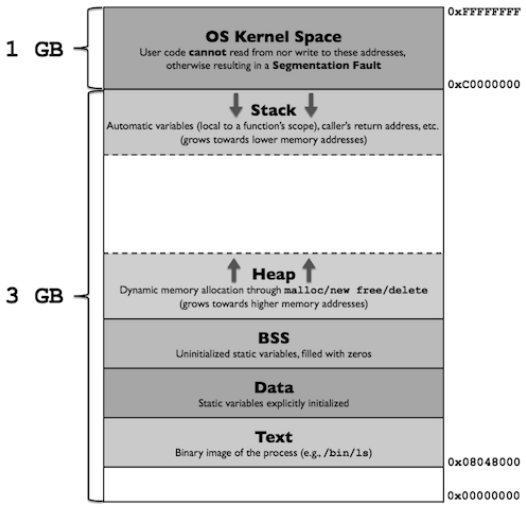
Sistemele de operare sunt sisteme de programe care oferă programatorilor o interfață simplă prin care să lucreze cu componentele complexe ale calculatorului, pe care le are în gestiune.

Fiind un software fundamental, sistemul de operare rulează într-un mod privilegiat, numit mod nucleu, în care are acces deplin la hardware și poate executa orice instrucțiune de care calculatorul e capabil. Pe de altă parte, celelalte programe rulează în modul utilizator, în care drepturile sunt limitate.

2.1.4.1 Procese și Fire de Execuție

Un program din modul utilizator, aflat în execuție, se numește **proces**. Asociat fiecărui proces este un **spațiu de adrese**, adică o zonă de memorie în care programul poate să scrie și să citească și care conține zone dedicate pentru programul executabil, datele programului, stiva acestuia. Pe lângă cele menționate, mai include regiștrii salvați ai procesorului, o listă de fișiere deschide, procese înrudite și alte informații necesare execuției.

Sistemele de operare moderne permit, prin intermediul unui concept numit **multiprogramare** (engl. "*multitasking*"), rularea a mai multor procese în același timp, cu condiția ca numai un număr fix (egal cu numărul de procesoare al calculatorului) să se afle în execuție. În acest timp, restul proceselor se află într-o stare de suspendare temporară, datele lor fiind salvate într-un **tabel de procese**, ce este un vector de structuri specifice (engl. "*process control block*"), una pentru fiecare proces. După o anumită cuantă de timp, stabilită de un proces dedicat numit **planificator**, procesele aflate în execuție sunt întrerupte, iar o altă serie de procese încep să fie executate.



Imagine 2.4: Organizarea spațiului de adrese al unui proces[4]

Fiecare proces își poate defini în cadrul său mai multe **fire de execuție**, cărora li se poate aloca timp de procesor de către planificator. Aceste entități împărtășesc același spațiu de adrese cu procesul care le-a creat, diferența fiind pentru stivă și pentru variabilele statice, ce sunt specifice și salvate într-un spațiu dedicat, local firului. Toate aceste spații aparțin de un bloc de mediu (engl. "*thread environment block*").

2.1.4.2 Apeluri de Sistem

Sistemul de operare expune programelor din spațiul utilizator o serie de funcționalități, ce vor fi executate cu privilegii depline, adică în modul privilegiat. Aceste **apeluri de sistem** se află în evidența sistemului de operare fie prin metode exportate de către o librărie, precum `ntdll.dll` de pe sistemele de operare Windows, fie printr-un index într-o tabelă, cum este cazul în sistemele de operare UNIX.

Pentru invocarea unui apel de sistem, procesul trebuie să pregătească datele ce vor fi transmise și să creeze o cerere de întrerupere, pentru a ceda contextul rutinei privilegiate. După ce aceasta se execută, va salva rezultatul într-o locație uzuală (registru sau zonă de memorie dată ca argument), va restabili contextul și va reda dreptul de execuție procesului apelant.

[4]https://gabrieletolomei.wordpress.com/miscellanea/operating-systems/in-memory-layout

2.1.5 Sisteme de Fișiere

Asemenea procesoarelor ce sunt abstractizate sub formă de procese, memoria este tratată în aceeași manieră. Ea este modelată de către sistemul de operare sub formă de **fișiere**, unități logice de informație create de procese. Această abordare are o serie de avantaje:

- Dimensiunea mare a informației ce poate fi stocată;
- Persistența informației, chiar și după terminarea procesului; și
- Concurența accesării informației, de către mai multe procese.

Toate fișierele sunt gestionate printr-o metodologie specifică, ce stabilește cum sunt structurate, accesate, modificate și protejate fișierele și ce este implementată într-o componentă a sistemului de operare numită **sistem de fișiere**.

2.1.5.1 Fișiere

Fișierele sunt cea mai mică unitate care permite salvarea persistentă de date.

Acestea sunt create și ulterior identificate de către procese prin intermediul unui **nume** ce conține o secvență dintr-un set de caractere, diferit de la o implementare la alta. În unele sisteme de fișiere, numele sunt compuse din două părți, separate prin punct. În timp ce prima identifică fișierul, **extensia** (cea de-a doua parte) indică anumite caracteristici ale fișierului, caracteristici ce pot fi folosite pentru procesarea lui.

În cazul în care extensia nu este prezentă, sistemele de operare precum cele UNIX se folosesc de o serie de octeți din antet, numită **număr magic** (engl. "*magic number*"), pentru a identifica tipul fișierului.

Pe lângă nume și date, sistemele de operare mai asociază fișierelor date suplimentare, numite **atribute**. Acestea pot oferi informații utile, precum timpul creării sau a ultimei modificări a fișierului, dimensiunea și drepturi necesare pentru a efectua diverse operații asupra sa.

Cele mai întâlnite operații asupra fișierelor, în sistemele de operare moderne, sunt: creare, redenumire, ștergere, deschidere, închidere, citire, scriere și modificarea atributelor.

2.1.5.2 Directoare

Fișierele descrise anterior sunt salvate, de regulă, în interiorul unor **directoare**. Acestea sunt o formă specială de fișiere, identificate prin nume, care pot avea copii sub forma unor referințe către alte fișiere. În acest fel, se construiește o structură arborescentă.

Pentru a prelua numele unui nod dintr-o astfel de structură, sunt disponibile două abordări: una **absolută**, în care se efectuează o parcurgere de la **directorul rădăcina** și până la nodul de interes, și una **relativă**, față de **directorul curent**.

Operații uzuale asupra directoarelor sunt: creare, redenumire, ștergere, deschidere, închidere, referențiere (prin care fișiere poate apărea în două directoare în același timp) și dereferențiere.

2.1.6 Fișiere Executabile

Atunci când sistemul de operare dorește să creeze un proces nou (din motive precum apeluri de sistem din cadrul altor procese, comenzi ale utilizatorului sau sarcini periodice), se folosește o componentă specială de încărcare (engl. "*loader*") pentru a pleca de la un **fișier de tip executabil**. După alte câteva operațiuni, printre care identificarea anumitor zone de memorie și popularea stării inițiale, procesul este rulat plecând de la un anumit punct, indicat în fișier printr-un câmp numit **punct de intrare** (engl. "*entry point*").

În același timp, fișierul executabil se poate folosi de resurse externe, nonvolatile, ce pot fi împărtășite de către mai multe programe și ce sunt numite **biblioteci**. Acestea conțin implementări ale unor comportamente predefinite, ce pot fi legate în două maniere. **Legarea statică** presupune copierea directă a lor în fișierul executabil final, în timp ce **legarea dinamică** doar referențiază acele implementări, urmând ca la încărcarea în memorie a programului să fie încărcate și acele librării, pentru a putea apela funcționalitățile ei.

2.1.6.1 Divizarea Spațiului de Adrese

Spațiul de adrese folosit de către un program este divizat în mai multe **segmente**, fiecare cu o funcționalitate diferită. Acestea sunt identificate prin etichete, ce apar și în limbajul de asamblare, pe baza căruia executabilul este format.

Segmentul de **cod**, având eticheta `.code`, conține codul mașină rezultat în urma compilării codului sursă al programului. Este de obicei protejat împotriva scrierii pentru a asigura anumite cazuri în care acest segment este suprascris neintenționat (din cauza unei referințe neinițializat corect) sau intenționat (în programe împachetate).

Segmentele de **date** conțin variabile globale, statice, constante sau externe. Acestea pot fi inițializate, caz în care segmentul are eticheta `.data`, sau neinițializate, într-un segment cu eticheta `.bss`. Pe de altă parte, segmentul de ***heap*** conține variabilele alocate dinamic, de exemplu prin apelul funcției `malloc`, specifice limbajului de operare C, din librăria `stdlib.h`.

Segmentul de **stivă** salvează, pe lângă argumentele programului, adresa de întoarcere (următoarea instrucțiune a funcției apelante, după apelul funcției curente), vechea referință către baza stivei, argumentele și variabilele locale. Toate acestea sunt grupate într-o zonă continuă de memorie numită **cadru** și identificată prin două referințe, unul către bază și unul către vârf.

Modul în care se gestionează stiva este numit **convenție de apelare** și este restricționat fie de către ISA, fie de către compilatorul cu care fișierele executabile se creează. Această convenție conține detalii precum modul în care

se transmit parametrii de la funcția apelantă la cea apelată, locația în care se salvează valoarea de retur și funcția care restaurează stivă.

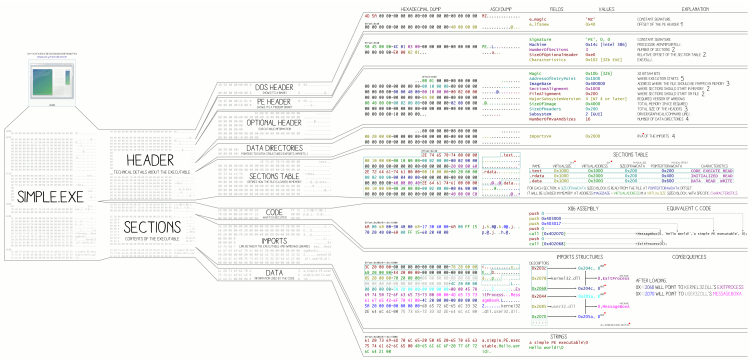
2.1.6.2 Portable Executable

Un format de fișiere reprezentativ pentru cele executabile, obiect și librării dinamice este **Portable Executable**, prezent pe sistemele de operare Windows.

Pentru formatul executabil, **antetul** începe cu un șir de octeți, specifici antetului de Microsoft Disk Operating System și păstrați datorită dorinței de compatibilitate. Este urmat de o semnătură de 4 octeți (`PE0\0\0`) și alte informații esențiale rulării programului, precum:

- Dimensiunea secțiunilor;
- Dimensiunile maxime ale secțiunilor de stivă și de *heap*;
- Adresa punctului de intrare, relativă la adresa de bază; și
- Subsistemul folosit.

După acestea, urmează un tabel, în care fiecare intrare reprezintă un antet al unei secțiuni, cu detalii precum dimensiunile fizice, virtuale și caracteristicile (o serie de biți ce indică anumite proprietăți ale secțiunii, printre care alinierea și drepturile asupra ei), și secțiunile fișierului executabil.



Imagine 2.5: Structura formatului Portable Executable[5]

2.1.6.3 Fișiere OLE

Formatul OLE, al cărui nume provine din abrevierea sintagmei din limba engleză "*Object Linking and Embedding*", este o tehnologie proprietară companiei Microsoft care permite încorporarea și referențierea de obiecte în cadrul unor

[5]https://github.com/corkami/pics/blob/master/binary/pe101/pe101l.png

fișiere compuse. De regulă, este utilizat în **suita de programe Microsoft Office**, pentru documente, prezentări, tabele de calcul și baze de date, și în mesajele Outlook.

Sunt disponibile două formate diferite OLE, și anume `1.0` și `2.0`. Ultimul este cel recomandat pentru a fi folosit când se creează documente de tip container[6], versiunea `1.0` fiind suportată de programele actuale numai pentru compatibilitate.

Fiind un obiect compozit, un fișier în format OLE este compus din:

- ***Fluxuri de date*** (engl. "*streams*"): Sunt asemănătoare unor fișiere incluse într-o arhivă. Fiecare este identificat printr-un nume, de exemplu fluxul `WordDocument` dintr-un fișier Microsoft Word care conține textul său.

- ***Depozite*** (engl. "*storages*"): Sunt grupări de fluxuri de date sau de alte depozite. Un exemplu de depozit este cel identificat prin numele `Macro`, ce conține acele secvențe de cod Visual Basic for Applications (abreviat VBA) folosite pentru automatizarea conținutului documentului.

- ***Proprietăți***: O proprietate este o valoare ce poate fi folosită la stocarea unei informații relevante, precum date despre documentul curent (titlu, autor, date de creare și modificare).

2.1.7 Comunicații în Rețea

Deși unele programe executabile sunt concepute pentru a rula pe un singur sistem de calcul, intervin situații în care este necesară o conexiune a mai multor astfel de sisteme. Acestă comunicație dintre ele, numită **rețea**, presupune trimiterea unor **pachete** cu date, date din cele ale resursei ce se dorește a fi partajată. În același timp, emițătorul și receptorul respectă un set comun de reguli, definit într-un **protocol**.

În proiectarea modernă, protocoalele de rețea sunt stratificate pentru a forma o **stivă**. Cea mai cunoscută este **stiva TCP/IP**, în care pe straturile de rețea și de transport apar protocoalele Internet Protocol (abreviat IP), respectiv Transmission Control Protocol (abreviat TCP). Tot sistemul global de dispozitive interconectate ce folosesc această suită de protocoale formează **Internetul**.

2.2 Aplicații Malițioase

Orice aplicație care este proiectată în mod intenționat pentru a provoca un prejudiciu unui utilizator, unui calculator sau unei rețele de calculatoare se numește **aplicație malițioasă** (engl. "*malware*").

[6] https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-oleds/fdc5e702-d09e-4344-a77f-eb079d41f23f

2.2.1 Tehnici, Tactici și Proceduri

De regulă, aplicațiile malițioase urmăresc un șablon comun. Primul pas surprinde momentul în care un utilizator face o acțiune ce provoacă descărcarea programului. Această acțiune poate consta în vizitarea unei pagini web malițioase, descărcarea unui atașament dintr-un email, a unui fișier dintr-o rețea de partajare sau conectarea unui stick USB cu un software (engl. "*firmware*") adaptat de către atacator. Odată ce aplicația malițioasă a fost rulată după descărcare, ea infectează calculatorul și începe să lucreze pentru îndeplinirea scopurilor creatorului.

Deși comportamentul acestor programe este diferit de la un exemplar la altul, s-a efectuat o standardizare prin intermediul **tacticilor, tehnicilor și procedurilor** (engl. "*tactics, techniques and procedures*" și abreviat TTP). Tacticile reprezintă motivul pentru care actorul efectuează acea acțiune malițioasă. Tehnicile, pe de altă parte, sunt modalitatea prin care obiectivul tactic este îndeplinit, în timp ce procedurile sunt implementări specifice ale programului malițios, utilizate în cadrul tehnicii.

Un *framework* public de cunoștințe despre programe malițioase, cu observații din lumea reală, este MITRE ATT&CK[7]. Conține matrice în care TTP-urile sunt grupate, permițând astfel analiștilor și organizațiilor să înțeleagă mai bine amenințările de care se lovesc.

2.2.2 Actori

Aplicațiile malițioase sunt realizate de către **actori**, ce pot fi clasificați în funcție de scopul efectuării unei astfel de activități prejudicioase:

- ***Criminalii cibernetici***: Reprezintă acele persoane care urmăresc obținerea de profit și de reputație, formând cea mai întâlnită categorie de actori. Ei pot fi individuali sau parte dintr-o echipă mai mare. Principalele tehnici folosite de ei sunt atacurile de tip *phishing* (obținerea de informații senzitive prin deghizarea sub o entitate de încredere), ingineria socială, atacul prin forță brută al parolelor și *ransomware*.

- ***Persoanele din interior***: Sunt angajați, contractori sau parteneri ai unei companii, care au acces la rețeaua, sistemele și date corporative. Dorința lor de a provoca daune companiei țintă se motivează prin dorința de a avea câștiguri financiare sau prin căutarea răzbunării. Se folosesc de tehnici precum exfiltrarea de date și abuzarea privilegiilor deținute.

- ***Actorii statali***: Pot fi parte din organizația internă a unui stat sau pot primi direcții, finanțare și asistență tehnică de la state. Ei atacă și obțin persistență în sectoarele publice și private cu scopul de a fura, schimba sau distruge informație. Cele mai întâlnite tehnici sunt atacurile de tip *spear-phishing* (atac de tip *phishing*, vizat unui anumit individ sau

[7] https://attack.mitre.org

unei organizații anume), ingineria socială și accesul de la distanță prin programe de tip *backdoor*.

- ***Hacktivists***: Cuprinde acei actori ce sunt motivați politic, social sau ideologic și ce întreprind acțiuni ilegale, malițioase pentru obținerea vizibilității sau a unor schimbări în ariile în care ei activează. Folosesc tehnici precum atacurile de întrerupere distribuită a serviciilor (engl. "*distributed denial of service*"), *doxing* (identificarea și publicarea online a unor informații) și vandalizarea website-urilor (engl. "*website defacement*").

- ***Organizațiile teroriste***: Folosesc acțiuni ofensive cu scopul de a obține câștiguri materiale, de a spiona sau de face propagandă. Tehnicile folosite sunt deteriorarea website-urilor și publicarea de informații false.

2.2.3 Taxonomia Programelor Malițioase

O clasificare mai grosieră decât cea prin atribuirea de TTP-uri constă în identificarea comportamentului unitar al programului malițios. Cum majoritatea programelor de acest tip cad într-una din aceste categorii, abordarea de față poate ajuta la efectuarea analizei prin stabilirea unei ipoteze ce poate fi confirmată sau infirmată în urma unei analize mai amănunțite:

- ***Backdoor***: Sunt programe ce se instalează singure, fără interacțiunea specifică a utilizatorului și cu scopul de a oferi atacatorilor acces și posibilitatea de a executa comenzi.

- ***Botnet***: Reprezintă un program prin care sistemele compromise primesc toate aceeași comandă de la atacator, prin intermediul unui server de comandă și control.

- ***Downloader***: Permite numai descărcarea și instalarea altui program malițios.

- ***Spyware***: Colectează informații de pe calculatorul infectat și le trimite către atacator.

- ***Rootkit***: Sunt programele specializate în ascunderea existenței altor programe malițioase pe sistemul local, de regulă față de programele de tip antivirus.

- ***Scareware***: Au scopul de a determina proprietarul sistemului infectat să cumpere anumite servicii, cu unicul obiectiv de a îmbunătăți situația financiară a atacatorului.

- ***Virușii*** (sau programe de tip ***worm***): Se propagă în rețea, infectând calculatoarele conectate între ele.

2.2.4 Indicatori de Compromitere

Indicatorii de compromitere reprezintă o evidență obținută din urma unei analize, care indică o posibilă infecție a unui sistem sau a unei întregi rețele. Aceste probe ajută analiștii și administratorii de sistem să detecteze încercările de intruziune și alte activități malițioase, anormale, ce se petrec. În același timp, pot fi partajați public sau între organizații (de exemplu, între o corporație și o companie de *threat intelligence* contractantă) pentru îmbunătățirea răspunsului în fața amenințărilor sau a strategiilor de remedierea a daunelor.

Cum acești indicatori sunt împrăștiați în tot sistemul, o analiză manuală a lor este ineficientă. Se folosesc programe automate, ce ușurează colectarea și identificarea maliției lor.

Cei mai întâlniți indicatori de compromitere se bazează pe:

- Fișiere, aplicații sau procese necunoscute;
- Fișiere standard sau regiștrii modificați;
- Trafic neobișnuit din rețeaua corporativă;
- Fișiere de jurnalizare ce conțin evidențe ale repetării unei anumite operațiuni (de obicei, în cadrul unui atac prin forță brută asupra parolelor); și
- Acțiuni suspicioase ale administratorilor sau ale conturilor privilegiate.

2.2.5 Tehnici de Analiză

În cadrul analizei programelor malițioase, ce este efectuată de personalul specializat, apar două abordări fundamentale și diferite, ce pot fi însă și întrepătrunse pentru o calitate superioară a procesului de investigație.

2.2.5.1 Analiză Statică

Analiza statică presupune examinarea aplicației malițioase fără a o executa.

O **analiză de bază** permite vizualizarea unor detalii generale despre executabil, fără a inspecta însă instrucțiunile ce vor fi executate la *runtime*. Deși să confirme maliția unui fișier și oferi informații despre funcționalitate (unele detalii pot fi folosite pentru crearea de semnături de sistem și de rețea), are dezavantajul că este inutilă în cazul programelor malițioase sofisticate, ce dispun de împachetare, putând rata comportamente importante.

Pe de altă parte, **analiza avansată** constă în ingineria inversă a executabilului și înțelegerea funcționalității lui din instrucțiunile pe care procesorul este delegat să le execute. Dificultatea ei este mai ridicată din cauza necesității unor cunoștințe de limbaj de asamblare, constructe de cod și de sisteme de operare.

Instrumentele avansate, care sunt folosite numai în al doilea tip de analiză statică, sunt **dezasambloare**. Constau în programe care translatează limbaj mașină, ce se regăsește în secțiunea de cod a fișierului executabil, în limbaj de

asamblare, ce poate fi mai ușor de interpretat de un specialist. Apar două tipuri de algoritmi de dezasamblare, care diferă prin modul de abordare al problemei:

- ***Dezasamblarea liniară***: Iterează fiecare bloc de cod și translatează fiecare instrucțiune găsită în echivalentul său în limbaj de asamblare. Este cel mai ușor de combătut de către programele malițioase întrucât ele pot să apeleze la simple tehnici de anti-dezasamblare pentru a ascunde instrucțiunile ce sunt executate, de fapt, de către procesor.
- ***Dezasamblarea orientată pe flux***: Examinează fiecare instrucțiune și construiește liste de locații ce urmează a fi dezasamblate. Abordarea aceasta este folosită în dezasambloarele moderne datorită acurateței rezultatelor obținute.

2.2.5.2 Analiză Dinamică

Analiza dinamică implică, spre deosebire de cea statică, execuția programului malițios într-un mediu controlat și observarea comportamentului său, cu scopul de a obține un mecanism de ștergere a sa de pe un sistem infectat sau de a produce semnături eficiente ce pot fi folosite pentru prevenirea infectării.

Analiza de bază numai rulează programul și observă efectele sale asupra sistemului, la nivel de sistem de fișiere, procese, regiștrii și rețea. De obicei, implică utilizarea unui ***sandbox***, care are însă dezavantaje precum confidențialitatea scăzută a documentelor scanate, detecția mașinii virtuale de către programul analizat și necesitatea oferirii de argumente în linie de comandă.

Versiunea **avansată** a analizei dinamice presupune folosirea unui **depanator**, ce se folosește de *breakpoint*-uri software (prin suprascrierea instrucțiunilor cu codul de operație `oxCC`, ce provoacă o întrerupere pentru sistarea execuției) sau hardware (prin regiștrii de control ai procesorului) pentru a executa instrucțiune cu instrucțiune programul.

Categorie	Tip de Analiză	Exemple de Instrumente
antivirus	de bază	Bitdefender Antivirus Plus, Kaspersky Total Security
calculator de *hash*	de bază	md5deep, WinMD5
motor de căutare specializat	de bază	VirusTotal
identificator de șiruri de caractere	de bază	Strings
decomprimator	de bază	PEiD, upx
identificator de funcții implementate, importate și exportate	de bază	DependencyWalker, PEView
identificator de resurse	de bază	Resource Hacker
analizor pentru antetul formatelor executabile	de bază	PE Browser Professional
dezasamblor	avansată	IDA Pro, Ghidra
librărie pentru automatizarea dezasambloarelor	avansată	IDAPython, Ghidra API, Capstone, pefile

Tabel 2.1: Categorii de instrumente pentru analiză statică

Categorie	Tip de Analiză	Exemple de Instrumente
sandbox	de bază	GFI Sandbox, Joe, Comodo
monitor de procese	de bază	procmon
exportator pentru regiștrii	de bază	RegShot
emulator de rețea	de bază	Apate DNS, INetSim
monitor pentru rețea	de bază	Wireshark
depanator la nivel utilizator	avansată	OllyDbg, WinDbg
depanator la nivel de nucleu	avansată	WinDbg, SoftICE
emulator	avansată	QEMU, Unicorn
librărie pentru automatizarea depanatoarelor și emulatoarelor	avansată	Qiling Framework

Tabel 2.2: Categorii de instrumente pentru analiza dinamică

2.3 Inteligență Artificială

Inteligența artificială reprezintă un domeniu al științei calculatoarelor care se ocupă cu realizarea de sisteme de calcul inteligente. Acestea din urmă sunt capabile, prin intermediul unor algoritmi dedicați, să rezolve probleme numeroase din lumea reală, specific umane, precum raționarea, reprezentarea cunoștințelor, planificarea și procesarea de limbaj natural.

2.3.1 Învățare Automată

O altă sarcină pe care acest tip de mașini inteligente o rezolvă este învățarea, subdomeniu al inteligenței artificiale cunoscut ca **învățare automată**. Prin intermediul acestei științe, se construiesc mașini capabile să învețe autonom rezolvarea unei game de probleme, fără a fi programate explicit în această direcție.

Înainte de aplicarea efectivă a algoritmului de acest tip, apar încă trei pași

premergători. Primul este de **creare a setului de date**, în care se adună din diferite surse exemplare etichetate sau neetichetate. Următorul este de **extragere a atributelor** din exemplare, obținându-se caracteristici specifice fiecărui exemplar. Acestea sunt însă în formă brută, formă care uneori este imposibil de procesat de către un algoritm de învățare automată, de unde și nevoia de **preprocesare**, etapă în care atributele brute sunt transformate într-o formă favorabilă învățării.

Generic, acest tip de algoritmi au ca intrare un **set de date** conținând **exemplare**. După finalizarea procesului de învățare, pe baza modelului de învățare automată (în continuare numit **model**) ce a fost **antrenat**, se pot realiza **predicții**.

2.3.2 Taxonomia Algoritmilor de Învățare Automată

În funcție de modul lor de funcționare, **algoritmii de învățare automată** pot fi categorisiți în:

1. ***Învățare supervizată***: Învățarea supervizată presupune prezența unui **set de date** $\{(x_i, y_i)\}_{i=1}^{N}$ cu N **exemplare** descrise prin intermediul unui **vector de atribute** x_i. În același timp, ele sunt etichetate, având atașate **etichete** y_i numerice sau categorice, care se încearcă a fi prezise. Astfel, pentru un nou vector de atribute x, algoritmul deduce probabilistic ce etichetă y îl poate reprezenta cel mai bine. Exemple aici sunt **regresia**, unde eticheta de prezis este una numerică reală, într-un spațiu continuu de valori, și **clasificarea**, unde etichetele sunt discrete, împărțind exemplarele în categorii (**clase**) diferite.

2. ***Învățare nesupervizată***: Acest tip de învățare presupune existența unui set de date neetichetate, $\{x_i\}_{i=1}^{N}$, din care algoritmul poate deduce informații utile rezolvării unui anumit tip de probleme.

3. ***Învățare semi-supervizată***: Aspectul semi-supervizat, prezent pentru acest tip de învățare, indică apariția ambelor tipuri de exemplare, atât etichetate, cât și neetichetate. În ciuda faptului că numai primul tip de exemplare este cel mai util, întrucât ele relevă concret asocierea dintre anumite caracteristici și o anumită etichetă, algoritmii din această categorie pot învăța din exemplarele neetichetate aspecte utile predicției, precum distribuția de valori pe care o poate avea un atribut anume.

4. ***Învățare de consolidare*** (engl. "*reinforcement learning*"): Mașina proiectată este pusă într-un mediu real sau digital, fiind constrânsă de limitările acestuia și unde este capabilă de a lua decizii. Pe de altă parte, starea în care se află este percepută ca un vector de proprietăți, vector ce este modificat o dată cu deciziile luate. Scopul tipului de învățare este ca sistemul inteligent să determine un comportament optim pentru a ajunge într-un punct dorit.

Toți acești algoritmi au la bază fie **modele**, ale căror parametrii sunt deduși pe baza exemplarelor, fie **instanțe**, caz în care modelul constă chiar în întregul set de date. Astfel, un model de regresie liniară va reține numai coeficienții dreptei/planului/hiperplanului care aproximează cel mai bine atributul țintă, în timp ce algoritmul celor mai apropiați K vecini va prezice prin examinarea atributelor și a etichetelor celor mai apropiate (având în vedere o distanță de dinainte stabilită) exemplare.

Un alt mod de a clasifica algoritmii de învățare automată este luând în considerare modul de deducere al parametrilor modelului. Algoritmii **superficiali** îi deduc chiar din proprietățile setului de date de antrenare. Cei **profunzi** folosesc o structură cu mai multe straturi, în care ieșirile dintr-un strat sunt conectate la intrările următorului strat, deci coeficienții find calculați nu neapărat pe baza proprietăților, ci pe baza ieșirilor din stratul anterior.

În cadrul lucrării, am folosit algoritmi de **învățare automată supervizată** datorită acompanierii fișierelor analizate de etichete care le definesc caracterul și care se doresc prezise. Pe de altă parte, ei sunt **bazați pe modele** datorită numărului mare de exemplare din seturile de date, care ar încărca memoria stației de lucru care folosește soluția software, cum ar fi fost cazul în care se construia o instanță. Metoda de învățare este una **superficială** întrucât antrenarea acestui tip de algoritmi este mai rapidă (ceea ce permite implementarea unei funcționalități de reantrenare), însă cu existența unui compromis în ceea ce privește acuratețea ce poate fi atinsă.

2.3.3 Algoritmi de Învățare Supervizată

2.3.3.1 Componență

Fiecare algoritm de învățare automată supervizată și bazat pe modele este alcătuit din **trei componente principale**. Prima este o **funcție de pierdere** care, prin intermediul parametrilor săi configurabili, reușește să aproximeze eticheta. Ea este optimizată cu ajutorul următoarelor două componente, și anume un **criteriu de optimizare** bazat pe funcția de pierdere și o **metodă de optimizare**, cu ajutorul căreia se verifică dacă parametrii iterației curente sunt mai adecvați decât cei din iterația anterioară. La finalul antrenării, se asigură faptul că algoritmul a ajuns într-un punct de minim local.

2.3.3.2 Evaluarea Performanțelor

Odată ce antrenarea unui model este finalizată, urmează o etapă de **testare**, în care sunt **verificate performanțele sale**. Algoritmii supervizați de învățare automată au avantajul de a avea exemplare cu etichete $\{y_i\}_{i=1}^{N}$, care pot fi prezise prin etichete $\{\hat{y}_i\}_{i=1}^{N}$. Validitate etichetelor poate astfel sa fie verificată obiectiv.

În cazul regresiei, eticheta prezisă este una numerică, într-un spațiu continuu de valori. Câteva metrici pe baza cărora performanțele pot fi cuantificate și pe care le-am folosit în cadrul lucrării sunt următoarele:

1. ***Eroarea absolută maximă***: Definită ca $max(|y_i - \hat{y}_i|)$, reprezintă cea mai mare eroare absolută. Surprinde predicția cea mai slabă calitativ, pe care modelul a făcut-o.

2. ***Eroarea absolută medie***: Această eroare este $\frac{1}{N} \cdot \sum_{i=1}^{N} |y_i - \hat{y}_i|$. Utilitatea ei apare la compararea cu valorile pe care etichetele le poate lua, fiind pe aceeași scală valorică, și are avantajul de a păstra aceeași greutate erorilor, fiecare contribuind în mod egal la construirea valorii finale a metricii.

3. ***Rădăcina erorii medie pătratice***: Definită ca $\sqrt{\frac{1}{N} \cdot \sum_{i=1}^{N} (y_i - \hat{y}_i)^2}$, este o metrică ce păstrează scala valorică a atributelor și care indică variația standard a erorilor. Are dezavantajul de a fi sensibilă la exemplarele aberante (engl. "*outliers*") întrucât erorile mari pe care acestea le provoacă au un efect mult prea mare asupra valorii finale a metricii. Pe de altă parte, poate fi utilizată ca și criteriu de optimizare datorită caracterului său diferențiabil.

4. ***Scorul*** R^2: Acest scor este calculat ca fiind raportul dintre variația erorilor predicțiilor și variația etichetelor reale, cu ajutorul formulei $1 - \frac{\sum_{i=1}^{N} (y_i - \hat{y}_i)^2}{\sum_{i=1}^{N} (y_i - \bar{y})^2}$, unde $\bar{y} = \frac{1}{N} \sum_{i=1}^{N} y_i$. Cu cât el este mai mare, cu atât modelul reușește să producă valori care potrivesc variația etichetelor reale ale exemplarelor. De regulă, pentru modelele de învățare automată, ia valori între:

   (a) 0, pentru un model care prezice de fiecare dată valoarea estimată, și anume $\sum_{i=1}^{N} y_i \cdot P(y_i)$; și

   (b) 1, pentru un model care prezice de fiecare dată eticheta numerică exactă, caz ce este însă rar întâlnit în realitate.

Din cauza aspectului discret pe care îl au etichetele folosite în cadrul clasificării, metricile folosite pentru acest tip de algoritm sunt diferite față de cele de regresie. În continuare, le vom trece în revistă considerând o clasificare binară.

1. ***Matrice de confuzie***: Este o matrice în care sunt expuse erorile de clasificare pe care le face un algoritm de clasificare, prin folosirea claselor reale și a celor prezise pe post de linii și de coloane. Astfel, pentru fiecare celulă ce corespunde clasei reale C_i și celei prezise C_j, apare numărul de exemplare care au eticheta reală C_i și cărora le-a fost asignată automat eticheta C_j. Matricea de confuzie ajută la identificarea perechilor de clase pentru care modelul antrenat face confuzii. În cazul existenței numai a două clase (una pozitivă, una negativă), matricea are patru celule:

   (a) ***Adevărat pozitive*** (engl. "*true positives*" și notat TP): Sunt exemplare pozitive, clasificate ca fiind pozitive.

(b) ***Fals pozitive*** (engl. "*false positives*" și notat FP): Sunt exemplare negative, clasificate ca fiind pozitive.

(c) ***Adevărat negative*** (engl. "*true negatives*" și notat TN): Sunt exemplare negative, clasificate ca fiind negative.

(d) ***Fals negative*** (engl. "*false negatives*" și notat FN): Sunt exemplare pozitive, clasificate ca fiind negative.

2. ***Acuratețe***: Este calculată cu formula $\frac{TP+TN}{TP+FP+TN+FN}$, fiind raportul dintre numărul de exemplare corect clasificate și a tuturor exemplarelor. Indică măsura în care modelul reușește să clasifice corect exemplarele.

3. ***Precizie***: Precizia este $\frac{TP}{TP+FP}$, semnificând abilitatea clasificatorului de a nu clasifica un exemplar cu eticheta negativă ca fiind pozitiv. Ea scade cu fiecare exemplar pozitiv clasificat greșit.

4. ***Reamintire***: Fiind dată de formula $\frac{TP}{TP+FN}$, indică abilitatea modelului de a identifica toate exemplarele pozitive.

5. ***Coeficientul de corelație al lui Matthews***: Coeficientul de corelație este dat de formula $\frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}$, fiind identic cu cel al lui Pearson. Ia în considerare toate celulele din matricea de confuzie și este un scor normalizat, în intervalul $[-1, 1]$. Un clasificator aleatoriu va avea coeficientul 0, iar unul care atribuie corect etichetele exemplarelor va avea scorul 1.

2.4 Alte Noțiuni

Soluția software, a cărei implementare va fi detaliată în capitolul următor, a folosit și alte concepte decât cele deja prezentate, de analiză de programe malițioase și de algoritmi de inteligență artificială. Le vom prezenta succint în cele ce urmează.

2.4.1 Analiza Similarității

Unul dintre obiectivele platformei dezvoltate este de a stabili similaritatea dintre două fișiere, ce sunt privite în contextul analizei datelor ca exemplare. Astfel, similaritatea lor poate fi stabilită prin aplicarea unor **funcții de similaritate**, de tipul $s(x_1, x_2)$, care returnează o valoare numerică ce crește o dată cu similaritatea dintre vectorii de caracteristici ai exemplarelor, x_1 și x_2.

Funcțiile de similaritate pot fi:

1. ***Cosinus***: $s(x_1, x_2) = \frac{x_1 \cdot x_2}{\|x_1\| \cdot \|x_2\|}$

2. ***Cosinus ajustat***: $s(x_1, x_2) = \frac{(x_1 - \|x_1\|) \cdot (x_2 - \|x_2\|)}{\|x_1 - x_m\| \cdot \|x_2 - x_m\|}$, unde x_m este vectorul medie al tuturor exemplarelor

3. ***Pearson***: Definită ca $s(x_1, x_2) = \frac{(x_1 - \|x_1\|) \cdot (x_2 - \|x_2\|)}{\|x_1 - \|x_1\|\| \cdot \|x_2 - \|x_2\|\|}$, această metrică de similaritate corespunde și coeficientului de corelație Pearson. Avantajul principal îl reprezintă abilitatea sa de a scăpa de tendința unor exemplare de a avea valori mai mari pentru toate caracteristicile considerate. Pe lângă aceasta, asemănător similarității cosinusului ajustat, evită situațiile în care o valoare mare a unui atribut are o pondere prea mare asupra valorii finale a similarității, prin scăderea mediei din vectorii de atribute.

2.4.2 Apel de Procedură la Distanță

Apelul de procedură la distanță (engl. "*remote procedure call*" și abreviat RPC) este folosit în soluția software pentru **comunicarea între un server lider la alte servere subordonate**. Reprezintă o tehnică ce generalizează noțiunea convențională de apel local de procedură prin faptul că procedura apelată nu trebuie să se afle în același spațiu de adrese cu procesul apelant. Astfel, cele două procese ce interacționează pot fi pe același sistem de calcul sau pe sisteme diferite, ce comunică în rețea prin intermediul unui protocol de comunicații.

Prin folosirea RPC, dezvoltatorii de aplicații distribuite sunt scutiți de detalii ce țin de interfațarea cu rețeaua. Independența de stratul de transport (ce le include și pe cele față de straturile de internet și de acces la rețea din modelul TCP/IP) face ca aplicația bazată pe RPC să fie compatibilă cu o varietate de protocoale.

Din punct de vedere al funcționării, RPC este asemănător apelului clasic de proceduri. Clientul apelează o procedură de pe server și așteaptă un răspuns, fiind suspendat până când acesta este primit sau până când cuanta maximă de timp de așteptare (engl. "*timeout*") este depășită. În acest timp, serverul procesează cererea și, la finalizare, returnează rezultatul sub formă de răspuns. Suspendarea menționată anterior poate fi evitată în cazul în care apelurile sunt efectuate în mod asincron.

2.4.3 Docker și Docker Compose

Docker reprezintă un set de produse de tip platformă ca serviciu care utilizează **virtualizarea la nivel de sistem de operare**. Am folosit această tehnologie pentru a oferi produsul software sub forma unui pachet, numit în acest context **container**.

Containerele pot fi definite ca unități standard de software care împachetează codul unei aplicații, împreuna cu toate dependințele de care aceasta are nevoie pentru a rula rapid și corect pe orice sistem de calcul. Spre deosebire de mașinile virtuale, care au beneficii similare în ceea ce privește izolarea și alocarea resurselor, containerele virtualizează sistemul de operare, și nu suportul hardware. Astfel, ele pot rula pe același sistem de calcul prin împărțirea nucleului de sistem de operare și prin rularea separată a proceselor din modul utilizator. Deși sunt izolate, containerele pot împărți zone din memoria nevolatilă, prin

intermediul unor volume, și pot comunica între ele, prin intermediul unor rețele definite local.

Containerele pot fi privite ca instanțe ale unor imagini, aflate în execuție. O **imagine** reprezintă un șablon cu instrucțiuni pentru crearea unui container. De obicei, imaginile sunt construite prin modificarea altor imagini, pe baza unui fișier text, `Dockerfile`. Acesta definește comenzile pe care un utilizator ar trebui să le ruleze în linie de comandă pentru a construi imaginea respectivă.

Toate containerele Docker rulează cu ajutorul unui program numit **Docker Engine**, ce poate fi setat pe o varietate de sisteme de operare, precum distribuții de Linux, macOS sau Windows Server. Acționează întocmai ca o aplicație de tip server-client, fiind compusă din următoarele componente:

- Un server cu un proces de tip daemon, `dockerd`, ce gestionează imaginile, containerele, rețelele și volumele;
- O suită de interfețe (engl. "*application programming interface*" și abreviat API) prin care se poate comunica și controla serverul; și
- O interfață în linie de comandă, `docker`.

Pe lângă Docker Engine, care folosește un container ca unitate de lucru, **Docker Compose** poate fi utilizat pentru definirea și rularea aplicațiilor multi-container. Pe baza unor fișiere de tip `Dockerfile`, în care se definesc imaginile necesare, și a unui fișier de configurare YAML, `docker-compose.yaml`, ce numește instanțele imaginilor definite anterior (containerele), Docker Compose poate construi, rula sau șterge o infrastructură de containere, specifică unei soluții software.

2.4.4 Python

Python este limbajul principal de programare, în care soluția software este implementată. Ca și caracteristici, este un **limbaj interpretat**, de nivel înalt și cu scop general. Suportă mai multe paradigme de programare, printre care cea procedurală (definirea de proceduri ce sunt apelate succesiv), cea orientată pe obiect (definirea de obiecte care încapsulează cod și date) și cea funcțională (definirea de funcții ce sunt aplicate simplu sau prin compunere).

Ocupând al doilea loc, împreună cu Java, în topul celor mai folosite limbaje de programare, Python poate fi folosit pentru o varietate de aplicații, de la analiza datelor și inteligență artificială și până la aplicații desktop și web.

Capitolul 3: Proiectarea, Implementarea și Testarea Platformei

Plecând de la informațiile expuse în capitolul anterior, de la cele referitoare la arhitecturile de procesoare și ajungând până la cele despre învățarea automată cu ajutorul unor algoritmi dedicați, am putut deduce cerințele software și arhitectura unui sistem în care inteligența artificială este folosită pentru a îmbunătăți procesul de analiză automată a programelor malițioase.

3.1 Cerințe Software

Pentru definirea **cerințelor software**, ne-am limitat la un context al organizațiilor mici și mijlocii, care doresc fie un strat suplimentar de securitate în infrastructura proprie, fie trierea unui flux de fișiere posibil malițioase, trimise spre a fi analizate manual. Această abordare favorizează deducerea nevoilor esențiale pentru o platformă de acest tip, putând ulterior finalizării acestei lucrări să studiem cum ele se pot scala unui cadru mai larg, pentru organizațiile de dimensiuni mari.

3.1.1 Cerințe Funcționale

Din punct de vedere al **cerințelor funcționale**, platforma trebuie să fie ușor de gestionat de către un **administrator de infrastructură organizațională**, fără ca acesta să aibă nevoie de cunoștințe specifice, în domeniile de analiză de programe malițioase și de ingineria datelor. Sunt permise acțiuni de **instalare a platformei** pe un server, de **dezinstalare** și de **actualizare**, cu păstrarea compatibilității cu versiunea anterioară (pentru comenzile interfeței de administrator și pentru API) și cu datele existente (configurație, seturi de date construite și modele antrenate).

Funcționalitățile pe care platforma trebuie să le asigure pot fi deduse din suita de utilizatori pe care o poate avea. În primul rând, o persoană având cunoștințele specifice, menționate anterior, va ocupa o poziție de **administrator al platformei**. Ea va folosi arhitectura de servere pentru a **lucra cu**

exemplarele benigne și malițioase și pentru a **gestiona seturi de date și modele**. Pe de altă parte, clienții sunt reprezentați de:

- ***Simpli membrii ai organizației***: Ei **scanează fișiere** aflate în posesia lor (de exemplu, un fișier Word descărcat de pe Internet, pe stația de lucru) prin intermediul unui produs software accesibil lor.

- ***Membrii ai departamentelor de securitatea informației*** (inclusiv analiști de programe malițioase): Ei își formează o premisă pe baza rezultatului returnat de modelele antrenate. Acest lucru ajută la **luarea unei decizii rapide** în privința unui fișier intrat în rețeaua organizațională (de exemplu, de blocare la nivel de *firewall*) și la **stabilirea unei ipoteze** premergătoare unei analize manuale, amănunțite.

- ***Alte sisteme ale organizației***: Folosesc API-ul pentru **scanarea automată a fișierelor**. Un exemplu aici ar fi integrarea cu un server de *email*, pentru scanarea atașamentelor tuturor *email*-urilor primite de către angajați.

3.1.2 Cerințe Nefuncționale

Trecând la **criteriile** pe care le-am avut în considerare **în dezvoltarea și evaluarea platformei**, acestea sunt după cum urmează:

- ***Performanțe bune ale modelelor antrenate***: Modelele antrenate în cadrul platformei sunt calitative, conform unor metrici de evaluare specifice învățării automate, mai exact specifice regresiei și clasificării.

- ***Programare în Python***: Limbajul de programare folosit pentru implementarea platformei este Python. Motivul este reprezentat de avantajele pe care acesta le prezintă, dintre care putem aminti disponibilitatea pe o varietate de platforme (engl. "*cross-platform*"), orientarea pe obiecte, modularitate prin creare de pachete și de module și caracterul de sursă deschisă.

- ***Interfețe multiple, intuitive***: Soluția software dispune de interfețe multiple, în funcție de tipul de utilizator: grafică, în linie de comandă și programabilă. Numitorul lor comun este faptul că sunt intuitive, impunând o ușurință în utilizare.

- ***Arhitectură de tip lider-subordonat***: Cum duratele procesării fișierelor și a antrenării de modele sunt ridicate, o abordare secvențială a sarcinilor ar fi ineficientă. Din această cauză, platforma dispune de o infrastructură de tip lider-subordonat (engl. "*master-slave*") în care un server central va delega sarcinile consumatoare de timp unor servere capabile din punct de vedere al resurselor.

- ***Calcul paralel***: Duratele anumitor sarcini computaționale sunt ridicate, iar o secvențiere a mai multora, chiar și de același tip, ar duce la o prea mare întindere pe axa temporală. Profităm de procesoarele multiple ale calculatoarelor moderne pentru a efectua acest tip de operațiuni (de exemplu, predicția unui rezultat pentru un fișier nou) în paralel.
- ***Containerizare***: Platforma este containerizată cu ajutorul Docker, pentru a izola componentele majore ale platformei între ele și de sistemul de operare al gazdei.
- ***Configurabilitate***: Anumite funcționalități ale platformei, care dispun de parametrii pe baza cărora funcționează, pot fi configurate cu ajutorul scrierii în fișiere speciale, a utilizării unor comenzi dedicate sau a setării unor câmpuri din secțiunea de setări a unor pagini web.
- ***Securitatea informației***: Securitatea este asigurată prin autentificare. În plus, este asigurată criptarea comunicației între serverele implicate în funcționarea platformei, cât și între platformă și clienții acesteia.
- ***Jurnalizare***: Platforma dispune de o componentă de jurnalizare a evenimentelor, ceea ce asigură trasabilitatea celor întâmplate și o mai bună depanare a problemelor apărute în timpul execuției.
- ***Documentare***: Codul sursă al aplicației este documentat pentru a asigura un mediu facil dezvoltării cu sursă deschisă, cât și în cazul preluării responsabilității de implementare de către alt departament sau altă organizație.

3.2 Arhitectura de Ansamblu

3.2.1 Servere. Containere

Platforma este alcătuită din mai multe **servere**, fiecare ocupându-se de un set de sarcini care variază în funcție de tipul de clienți de care răspunde. Apar astfel trei tipuri de servere, pe care le vom descrie în cele ce urmează.

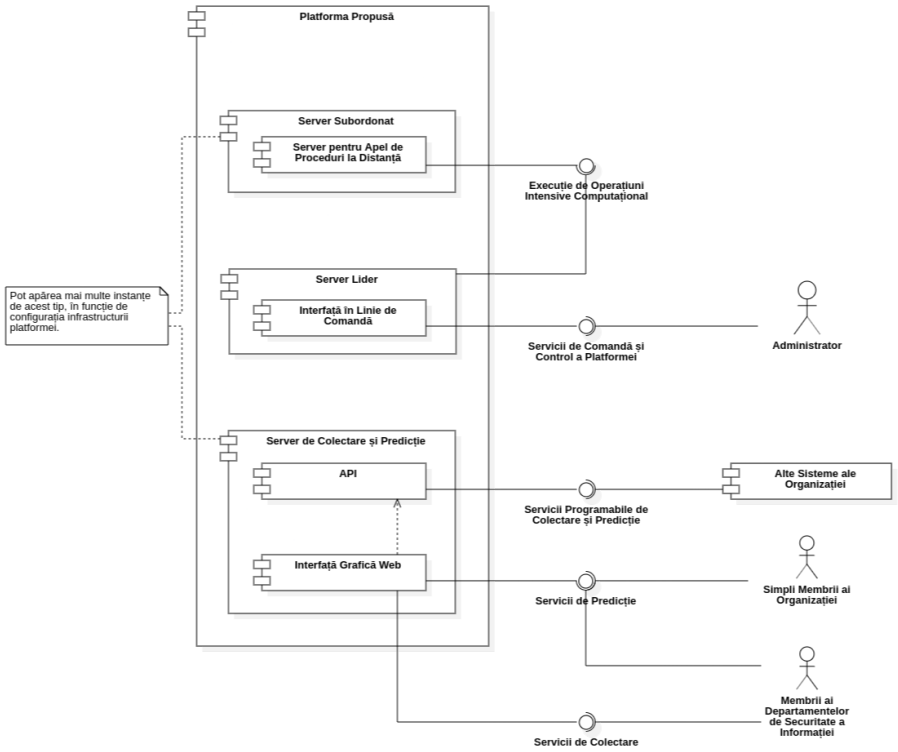
Primul tip de server este cel **subordonat**, ce este capabil din punct de vedere al resurselor alocate. El așteaptă delegări de operații pe care le poate executa, delegările ce sunt efectuate prin intermediul unor apeluri de proceduri la distanță, inițiatorul fiind serverul lider.

Serverul **lider** primește sarcini de la administratorul platformei, prin intermediul unei interfețe în linie de comandă. El verifică disponibilitatea serverelor subordonate pe care le coordonează (cu care are legături active) și deleagă sarcinile către unul dintre acestea, conform unui algoritm de planificare. Împreună cu tipul prezentat anterior, formează arhitectura de tip lider-subordonat.

Ultimul tip de server joacă, prin intermediul API-ului și a interfeței grafice web pe care le expune, un dublu rol: **de colectare și de predicție**. În primul rând, el primește de la clienții specifici, analiști sau alte sisteme ale

organizației, fișiere sau chiar atributele extrase pentru un anumit fișier, acompaniate de rezultate ale unei analize detaliate. Această pereche de informații ajută la îmbunătățirea modelelor antrenate prin intermediul procedeului de reantrenare. În al doilea rând, oferă servicii de predicție pentru clienții enumerați anterior, cât și pentru simpli membrii ai organizației.

Toate aceste tipuri de server sunt izolate individual în câte un **container** Docker, lucru care facilitează, printre altele, portabilitatea aplicației întrucât aceasta va rula într-un mediu standardizat, setat corespunzător, indiferent de sistemul de calcul gazdă. Pe de altă parte, se oferă posibilitatea de scalare a infrastructurii, în funcție de nevoile organizației, prin multiplicarea containerelor pentru serverele subordonate (de exemplu, în cazul unei antrenări intensive de modele) și pentru colectare și predicție (de exemplu, în cazul unui număr mare de clienți ce solicită servicii de publicare de rezultate exacte și de predicție pentru fișiere).



Imagine 3.1: Diagramă de componente, relativă la serverele implementate

Unele dintre funcționalitățile lor sunt comune, apărând în mai multe tipuri de server în același timp. De exemplu, funcționalitatea de extragere de atribute este necesară atât serverului subordonat, care o folosește pentru procesarea fișierelor ce compun un set de date folosit pentru antrenarea unui model, cât și celui de predicție, care o utilizează pentru determinarea atributelor, pas necesar în procesul de predicție. Astfel, serverele sunt dezvoltate prin importarea în cadrul implementării lor proprii a unor componente software reutilizabile, numite **module**.

3.2.2 Module

Drumul parcurs de datele utilizatorului în interiorul platformei, de la informații brute precum fișierele benigne și malițioase și până la modelele folosite pentru a crea predicții, poate fi privit ca o **linie de asamblare**. Astfel, un **set de module** capabile să proceseze date lucrează împreună la construirea graduală a unui rezultat. Fiecare dintre ele este conectat într-o manieră secvențială la ieșirea unuia, de la care preia date de intrare, și la intrarea altuia, căruia îi predă datele sale de ieșire.

Am dedus o serie de module specifice, pe baza cerințelor funcționale menționate anterior și a modurilor în care analiza de fișiere malițioase și antrenarea de modele de învățare automată funcționează. Modulele, cât și prezența lor în fiecare tip de server, sunt listate în tabelul de mai jos.

Nume	Server subordonat	Server lider	Server de colectare și predicție
Modul pentru etichetarea fișierelor și gestionarea seturilor de date	Da	Nu	Da
Modul pentru extragerea atributelor din fișiere	Da	Nu	Da
Modul pentru preprocesarea atributelor	Da	Nu	Da
Modul pentru gestionarea modelelor de inteligență artificială	Da	Nu	Da
Modul pentru gestionarea serverelor subordonate	Nu	Da	Nu

Tabel 3.1: Prezența modulelor în fiecare tip de server

3.2.2.1 Modul pentru Etichetarea Fișierelor și Gestionarea Seturilor de Date

Structura informațională de tip linie de asamblare are ca punct de plecare fișierele benigne și malițioase. Ele sunt identificate pe baza unei perechi de nume - tip (benign și malign) și compun setul de date principal, din care sunt selectate exemplare pentru crearea unor seturi de date folosite separat, pentru antrenarea unor modele. Fișierele provin din **alte surse de date**:

A. Set de date special creat pentru această platformă[1]

Setul de date special creat este compus, la rândul lui, din fișiere din surse publice:

- ***Setul de date "Malware Detection PE-Based Analysis Using Deep Learning Algorithm Dataset"***[2]: Conține fișiere executabile PE benigne și malițioase.
- ***Proiectul MalwareBazaar***[3]: Are scopul de partajare de programe malițioase cu comunitatea de securitate a informației. Din cadrul acestei surse am extras fișiere OLE malițioase.
- ***Motorul de căutare DuckDuckGo***[4]: L-am folosit pentru găsirea de fișiere OLE benigne, prin căutări aleatoare de documente (de exemplu, folosind șablonul `filetype:doc`).

Cum o procesare complet automată nu a fost posibilă, am efectuat manual o serie de pași, din linie de comandă, aceștia fiind însă replicabili prin intermediul unui script[5] realizat ulterior. Primul pas a fost cel de **descărcare**, în care toate fișierele din sursele enumerate anterior au fost salvate local, filtrate după extensie și mutate într-o structură standardizată de foldere. Fișierele au primit, în pasul de **redenumire**, un nume compus din *hash*-ul lor SHA256 și o extensie aferentă tipului (`.exe` pentru fișierele PE și `.ole` pentru cele OLE).

Ținând cont de numărul de fișiere ce trebuiau scanate automat și de limitele pentru numărul maxim de scanări ce puteau fi efectuate într-un anumit interval de timp, impuse de motorul VirusTotal[6], am preferat utilizarea unor servicii în *cloud* pentru pasul de **scanare** a fișierelor. Toate *hash*-urile lor au fost salvate într-un fișier ce a fost urcat într-un spațiu de stocare din Cloud Storage, soluție software oferită de Google Cloud Platform[7]. Acestea erau consumate, una câte una, de către un script implementat cu ajutorul Cloud Functions și declanșat prin intermediul unui planificator de tip Cloud Scheduler. Rezultatele,

[1] https://github.com/iosifache/DikeDataset

[2] Anh Pham Tuan et al. "Malware Detection PE-Based Analysis Using Deep Learning Algorithm Dataset". In: (2018). URL: `https://figshare.com/articles/dataset/Malware_Detection_PE-Based_Analysis_Using_Deep_Learning_Algorithm_Dataset/6635642`

[3] https://bazaar.abuse.ch

[4] https://duckduckgo.com

[5] https://github.com/iosifache/DikeDataset/blob/main/others/scripts/get_files.sh

[6] https://www.virustotal.com

[7] https://cloud.google.com

reprezentate de parți specifice ale rapoartelor de scanare cu motoare antivirus, au fost salvate într-un alt fișier. Am adăugat ulterior conținutul lui în cadrul platformei, unde rezultatele scanărilor au putut fi procesate automat pentru **etichetarea** fișierelor, proces pe care îl vom descrie mai jos.

B. Mecanism de publicare a unor rezultate de analize avansate

În cadrul structurii de foldere folosită, apare un folder special în care fișiere (sau chiar atribute deja extrase ale unor fișiere) pot fi **publicate**, împreună cu etichetele lor. Nevoia unui astfel de mecanism apare atunci când analiștii de programe malițioase examinează în detaliu un exemplar și doresc să ajute la îmbunătățirea performanței modelelor antrenate prin oferirea unor date de încredere, care să fie folosite în cadrul unui posibil proces de reantrenare.

Publicarea poate fi realizată prin intermediul interfeței grafice și, implicit, prin intermediul API-ului. Toate datele provenite din această sursă sunt considerate corecte, fișierele nemaifiind scanate cu acea componentă specifică ce apare în cazul sursei de date anterior detaliată.

Considerând componentele de care depinde învățarea automată, apare nevoia existenței unor **etichete** care ajută, prin natura și semnificația lor, modulul de inteligență artificială pentru predicția fiecăreia dintre ele:

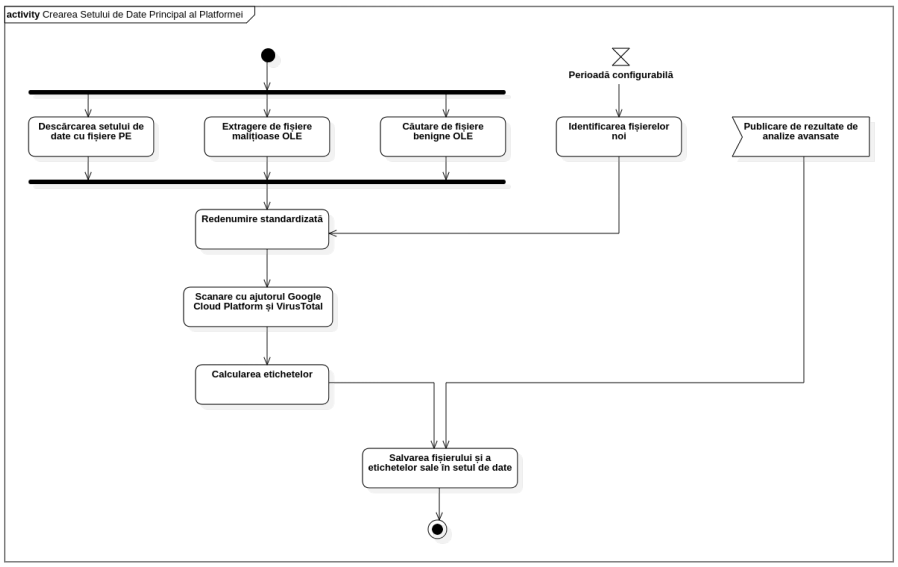
- ***Maliția***: Maliția este un indice numeric, ce indică probabilitatea ca un fișier să fie malițios. Valorile posibile sunt cuprinse între 0 și 1.

- ***Apartenența la familii de programe malițioase***: Acestea reprezintă, de asemenea, indici numerici pentru probabilitatea ca fișierul să aparțină de o anumită familie de programe malițioase. Valorile sunt cuprinse între 0 și 1, suma lor pentru un anumit fișier fiind egală cu 1. Familiile considerate sunt *Generic*, *Trojan*, *Ransomware*, *Worm*, *Backdoor*, *Spyware*, *Rootkit*, *Encrypter* și *Downloader*.

Răspunzător pentru **etichetarea fișierelor benigne și malițioase** este primul modul al platformei. Sistemul de foldere specific acestuia permite adăugarea manuală sau automată de fișiere noi în folderele specifice. Platforma **detectează exemplarele neavute în evidență**, prin verificarea periodică a locațiilor și a conținutului lor.

Modulul redenumește și scanează aceste fișiere, rezultatele scanărilor cu VirusTotal fiind procesate într-o manieră identică a acelora provenite de la scanarea în *cloud* a întregului set de date. Cu ajutorul unei medii ponderate asupra verdictului fiecărui motor antivirus, în care un vot pentru caracterul malign cântărește mai mult decât unul pentru caracterul benign, a fost determinată maliția. Etichetele atribuite de fiecare motor au fost folosite ca dovezi ale apartenenței la o anumită familie, ele fiind grupate și numărate, iar rezultatele normalizate pentru a obține probabilitățile finale.

De menționat este faptul că pentru fișierele presupuse benigne (mutate în folderul specific lor), pasul de scanare a fost sărit întrucât etichetarea este efectuată folosind valoarea 0 pentru toate etichetele considerate.



Imagine 3.2: Diagramă de activitate pentru crearea setului de date principal

Ultimul rol al acestei componente este de **a gestiona seturile de date** pe durata întregului lor ciclu de viață. Crearea unui set de date se face prin intermediul unor fișiere de configurare cu un format specific, ce cuprinde următorii parametri:

- ***Formatul fișierelor***: Formatul poate fi PE sau OLE.
- ***Numărul de exemplare incluse***: Este un număr întreg, notat în continuare $exemplare\_incluse$.
- ***Raportul dintre fișiere benigne și malițioase***: Este un număr real, cuprins între 0 și 1 și notat în continuare $raport\_benign\_malign$.
- ***Maliția minimă***: Este un număr real, cuprins între 0 și 1.
- ***Familii de programe malițioase***: În funcție de familiile incluse în setul de date, lucru care este setat în fișierul de configurare prin specificarea lor sub forma unei liste, fiecărui fișier malițios candidat i se calculează o sumă a probabilităților de apartenență la familiile selectate. Lista se ordonează descrescător și se selectează ulterior atâtea fișiere câte au fost solicitate, și anume $exemplare\_incluse * (1 - raport\_benign\_malign)$.

Pe baza acestor parametrii se stabilesc limite pentru caracteristicile exemplarelor și se filtrează setul de date principal al platformei. Dacă toate condițiile sunt verificate, are loc copierea detaliilor exemplarelor selectate din setul de date

principal într-un set de date nou, la care apare și o linie specială, de metadate, în care sunt reținute detaliile de configurare ale acestuia.

Seturile de date pot fi citite prin operațiunea de listare a tuturor (în care se accesează numai metadatele, nu și lista de exemplare incluse) sau prin cea de citire efectivă (care apare, de exemplu, în cadrul modulului specific modelelor). În plus, sunt posibile operațiuni de actualizare, prin intermediul operației de publicare a unor rezultate, și de ștergere a seturilor de date.

3.2.2.2 Modul pentru Extragerea Atributelor din Fișiere

Primul modul inclus în procesul de antrenare a unui model de inteligență artificială este cel de extragere a atributelor. Rolul său este de a procesa fișierele benigne și malițioase, **extrăgând din cadrul lor acele caracteristici specifice**, conform cu o configurație oferită de către administrator.

La nivel arhitectural, modulul este reprezentat de o componentă care gestionează aplicarea unor sub-componente numite **extractori**. Acestea din urmă sunt capabile de a procesa numai anumite tipuri de fișiere și sunt specializate pe anumite caracteristici ale formatului de care se ocupă.

Ieșirile modulului de extragere a atributelor sunt conectate la intrările modulului următor din structura informațională de tip linie de asamblare, cel care se ocupă de preprocesarea lor. Pentru a asigura păstrarea semnificației atributelor returnate de extractorii folosiți, acestui modul i se impune o anumită configurație.

Tipurile posibile de extractori sunt:

- `StaticStrings`: Fiind un extractor de tip static, iterează prin conținutul fișierului, la nivel de octet, pentru a detecta **șirurile de caractere printabile** cu anumite lungimi și număr de apariții minime, parametrii care sunt setați în configurația platformei. Ieșirea sa constă într-o listă de șiruri de caractere printabile.

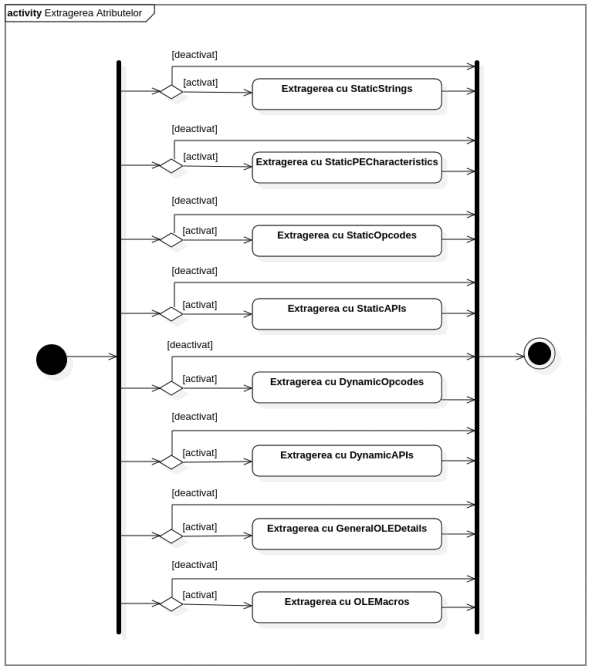
- `StaticPECharacteristics`: Neavând nevoie de rularea executabilului, acest extractor are ca ieșire **atribute specifice formatului de fișiere PE**, precum dimensiunea fișierului, funcții importate și exportate, librarii importate și detalii (nume, entropie, dimensiuni virtuale și brute) ale secțiunilor.

- `StaticOpcodes` și `StaticAPIs`: Extractoarele folosesc ca dezasamblor Ghidra pentru a efectua o analiză statică pentru extragerea unor liste cu toți **identificatorii codurilor de operație** găsite în cadrul executabilului analizat, respectiv cu toate **apelurile către Windows API**. Principalul dezavantaj al metodei folosite este că nu asigură aceeași credibilitate precum o analiză dinamică, în care se observă exact care sunt codurile de operație rulate sau apelurile de sistem efectuate. Astfel, în cazul unor constructe de cod repetitive sau a unor tehnici de dezasamblare (de exemplu, împachetare), extractorul nu reușește să producă o informație complet corectă.

- `DynamicOpcodes` și `DynamicAPIs`: Ele realizează o analiză dinamică, emulând executabilele într-un mediu controlat cu ajutorul unui *framework* numit Qiling[8]. Au același tip de ieșire precum extractorii descriși anterior și prezintă dezavantajul că biblioteca menționată nu implementează toate apelurile de sistem disponibile în Windows[9] și folosite de programele malițioase, ceea ce face ca extractorii aceștia să nu reușească să obțină rezultate utile pentru anumite fișiere executabile.

- `GeneralOLEDetails`: Acesta este un extractor care analizează static fișierul Office, extrăgând **detalii generale ale formatului OLE**. Ele variază de la text găsit în anteturi, număr de pagini și cuvinte, timpi de creare și modificare, indicatori (engl. "*flags*"), numărul de obiecte de tip Flash și de sectoare, cât și detalii (nume și dimensiune) ale intrărilor în directoare (engl. "*directory entry*").

- `OLEMacros`: Parcurge fișierul OLE în căutarea de ***macro***-uri VBA. Ieșirea sa este reprezentată de o listă cu codul sursă pentru fiecare *macro* identificat.



Imagine 3.3: Diagramă de activitate pentru extragerea atributelor

[8]https://qiling.io
[9]https://github.com/qilingframework/qiling/issues/751#issuecomment-822122311

3.2.2.3 Modul pentru Preprocesarea Atributelor

După ce toate atributele brute au fost determinate cu ajutorul modulului de extragere, ele sunt preluate sub formă matriceală (în care fiecare linie reprezintă un exemplar, iar coloanele tipuri de atribute) de modulul de preprocesare pentru **a le transforma în reprezentări mai adecvate învățării automate**.

Asemenea predecesorului său, acest modul aplică sub-componente de transformare, numite **preprocesoare**, specializate pe un anumit tip de atribute și selectate în funcție de ieșirile extractorilor. Fiecare preprocesor implementează operații specifice, pe care le aplică la nivelul atributelor de care este responsabil. De menționat este că premergător se asigură inexistența unor valori lipsă în setul de date, printr-un proces de **imputare** cu valori fixe, specifice tipului de date al atributului.

Preprocesoarele implementate sau importate din biblioteca `scikit-learn`[10] sunt:

- `Identity`: Replică datele de intrare la ieșire.
- `Binarizer`: Binarizează datele conform cu o limită stabilită.
- `KBinsDiscretizer`: Grupează date continue în intervale.
- `Counter`: Numără elementele componente ale unui atribut.
- `CountVectorizer`: Transformă un text într-o matrice de numere de apariții pentru fiecare cuvânt.
- `NGrams`: Numără grupuri de caractere consecutive, care apar într-o fereastră de o lungime oferită ca parametru. Cum pentru lungimi mai mari numărul de atribute devine mare, toate literele textului pot fi opțional transformate în minuscule și selectat un set de caractere mai restrâns.
- `GroupCounter`: Calculează, pentru fiecare grup, numărul de elemente incluse în el. Grupurile și elementele copii sunt precizate în configurația preprocesorului. În cadrul platformei, au fost definite grupuri pentru coduri de operații executate de procesor (de exemplu, `call` și `ret` sunt considerate operațiuni de lucru cu stiva) și pentru funcții exportate de Windows API (de exemplu, grupul de funcții criptografice include apeluri precum `CertOpenStore` și `CryptCreateHash`), ultimele fiind preluate din cadrul unei implementări[11] cu sursă deschisă a unui articol științific[12].
- `SameLengthImputer`: Aduce toate atributele exemplarelor la aceeași lungime, fie la una stabilită ca parametru, fie a exemplarului cu cea mai mare lungime.

[10] https://sklearn.org
[11] https://github.com/Hullgj/report-parser
[12] Gavin Hull, Henna John, and Budi Arief. "Ransomware deployment methods and analysis: views from a predictive model and human responses". In: *Crime Sci.* 8 (2019)

După ce toate preprocesoarele sunt aplicate, toate informațiile rezultate sunt de tip numeric. Ele sunt **scalate** în intervalul $[0, 1]$, unde pe valoarea 0 este proiectată valoarea minimă găsită în interval și pe 1 valoarea maximă. Acest procedeu asigură "*o viteză mai mare de învățare*" și că "*intrările se află într-un interval relativ mic, pentru a evita problemele pe care le au calculatoarele când lucrează cu numere foarte mari sau foarte mici*" [1].

Ultima operațiune efectuată în cadrul acestui modul, înainte ca datele să fie oferite următorului cu scopul de a se efectua o antrenare a unui model pe baza lor, este **reducerea dimensionalității**. Motivul este că algoritmii lucrează mai eficient (mai rapid și cu o acuratețe mai mare) atunci când procesează o "*informație de dimensionalitate redusă, care rezumă informația din spațiul inițial al atributelor*"[13].

Un exemplu relevant numărului mare de atribute la care se poate ajunge este cazul aplicării peste rezultatul unui extractor `StaticStrings` a unui preprocesor `NGrams`, având setate parametrul $N = 2$ și alfabetul englez de minuscule (cu lungime de 26 de caractere). Astfel, lungimea vectorului de atribute returnat este de $26^2 = 676$, dintre care multe valori sunt redundante (care sunt 0 pentru toate exemplarele analizate sau care au o variație infimă).

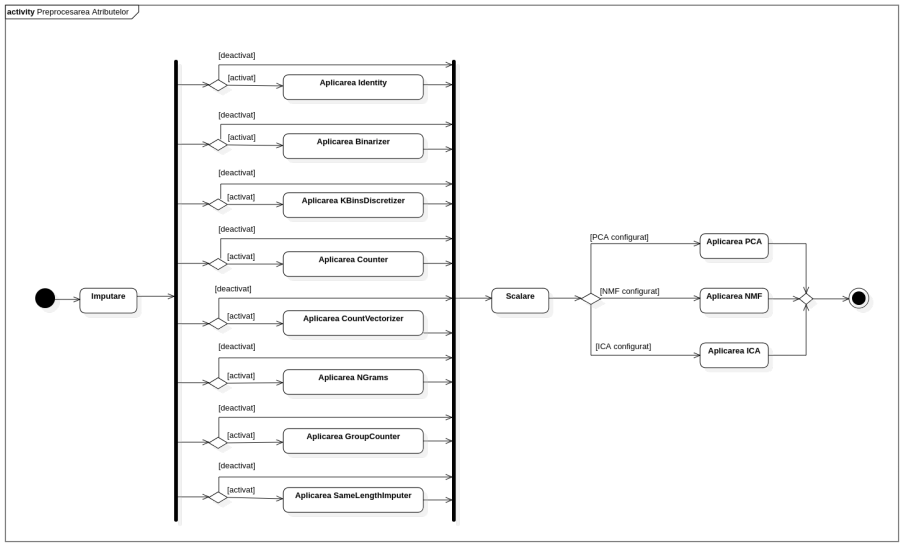
Pentru toți algoritmii de reducere a dimensionalității, numărul de atribute returnate este determinat în funcție de un parametru configurabil care indică variația minimă ce trebuie înglobată în toate acestea sau numărul total de atribute create. Algoritmii selectați sunt:

- ***Analiza Componentelor Principale*** (engl. "*principal component analysis*" și abreviat PCA): Presupune determinarea unei baze în care toți vectorii unitate sunt ortogonali și care reprezintă cel mai bine variația datelor. După, se transformă datele în această nouă bază și se selectează cele mai relevante atribute.

- ***Factorizarea în Matrice Non-negative*** (engl. "*non-negative matrix factorization*" și abreviat NMF): Găsește două matrice non-negative, de dimensiuni $m * o$ și $o * n$, a căror produs să aproximeze matricea non-negativa a atributelor, de dimensiuni $m * n$. Atributele relevante sunt chiar cele reprezentate de dimensiunea o a celor doua matrice găsite.

- ***Analiza Componentelor Independente*** (engl. "*independent component analysis*" și abreviat ICA): Găsește componentele independente din care fiecare atribut este presupun ca fiind compus.

[13] Alice Zheng. *Feature Engineering for Machine Learning: Principles and Techniques for Data Scientists*. O'Reilly Media, 2018



Imagine 3.4: Diagramă de activitate pentru preprocesarea atributelor

3.2.2.4 Modul pentru Gestionarea Modelelor de Inteligență Artificială

Atributele procesate sunt folosite în continuare ca date de intrare pentru **antrenarea unui model de inteligență artificială**.

Crearea unuia pleacă de la un fișier de configurare cu format special, în care sunt specificate și detaliile configurării modulelor de extragere și de preprocesare a atributelor astfel:

- ***Setul de date***: Indică ce set de date este folosit pentru antrenarea modelului.

- ***Obiectivul modelului***: Tipurile au fost stabilite prin preluarea celor mai întâlnite obiective din publicațiile de actualitate [3]. Ele indică de ce natură sunt datele de ieșire ale modelului. Primul obiectiv este cel de **regresie a maliției**, prin care, pe baza atributelor fișierului, i se prezice maliția. Următorul este cel de **clasificare**, prin care un fișier este încadrat probabilistic în mai multe familii de programe malițioase. Ultimul obiectiv, inclus automat în cele anterioare, este reprezentat de **analiza de similaritate**, prin care atributele exemplarului nou, trimis spre a fi analizat, sunt comparate cu atributele celor aflate în setul de date și sunt astfel selectate exemplarele care obțin cel mai mare scor de similaritate Pearson.

- ***Listă de extractori***: Pentru fiecare, apare și o listă de preprocesoare atașați.

- ***Algoritmul de reducere a dimensionalității***: Este numele de identificare al algoritmului folosit.

- ***Configurația algoritmului de reducere a dimensionalității***: Poate fi numărul caracteristicilor selectate de algoritmul de reducere a dimensionalității sau variația minimă înglobată de acestea (numai pentru PCA).

- ***Raportul dintre numărul de exemplare folosite pentru antrenare și a acelora din tot setul de date***: Acest număr întreg, între 0 și 1, este folosit la împărțirea setului de date folosit.

- ***Algoritmul de învățare automată***: Este numele de identificare al algoritmului folosit.

Algoritmii de învățare automată folosiți sunt unii de regresie. Aceștia se pretează atât pe regresia maliției, cât și pe clasificarea exemplarelor pe familii întrucât aceasta este cu etichete multiple și limite moi (engl. "*soft multi-label*"). Acest caracter presupune că se va returna, pentru fiecare clasă pentru care există etichete, o probabilitate de apartenență a fișierului analizat în ea, condiția fiind aici ca suma lor să fie 1. Algoritmii selectați aparțin de categoria supervizată și sunt:

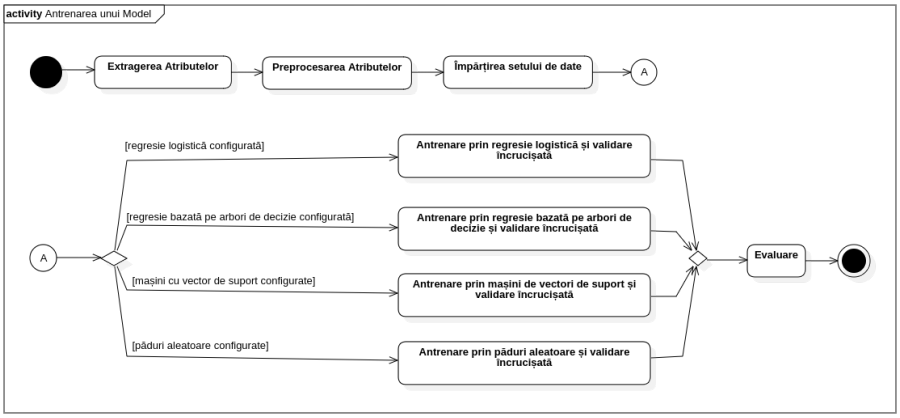
- ***Regresie pe bază de arbori de decizie***: Estimează un rezultat prin verificări repetate ale unor condiții până când modelul este suficient de încrezător pentru a face o predicție.

- ***Mașini cu vectori de suport*** (engl. "*support vector machines*"): Folosește conceptul de vectori de suport pentru determinarea unui hiperplan ce minimizează erorile.

- ***Păduri aleatoare*** (engl. "*random forests*"): Antrenează o multitudine de arbori de decizie și folosește pe post de predicție media predicțiilor individuale ale fiecărui arbore, cu scopul de a îmbunătăți acuratețea și de a controla supraînvățarea.

După ce fișierul este citit, are loc **divizarea în două a setului de date**, format din atributele procesate. Primul, cel de antrenare, este folosit numai pentru antrenarea modelului, în timp ce al doilea, cel de test, este folosit pentru evaluarea rezultatelor modelului selectat.

Antrenarea modelului se efectuează conform procedeului de **validare încrucișată**, pe setul de date specific, prin antrenări succesive de modele. Selectarea modelului final se face prin alegerea celui care obține cea mai mică eroare medie pătratică.



Imagine 3.5: Diagramă de activitate pentru antrenarea unui model

Antrenarea unui model declanșează o **evaluare avansată** a acestuia, pe baza unor seturi de criterii specifice obiectivului său. Pentru modelele de regresie a maliției, se determină eroarea maximă, eroarea medie, eroarea medie pătratică și scorul R^2. Pe lângă acestea, pentru toate predicțiile efectuate, se determină eroarea absolută și se grupează în cadrul unei histograme.

Din cauza caracterului modelele de clasificare utilizate în cadrul platformei, pentru fiecare etichetă specifică unei familii de programe malițioase, este oferită o valoare probabilistică într-un spațiu continuu. Astfel, este necesară o relativizare în ceea ce privește rezultatele prezise, pentru a putea aplica metodele clasice de evaluare la nivel de clasă (prin alegerea unei clase anume) și la nivel de valoare (prin stabilirea unei limite pentru binarizarea etichetelor returnate, peste care un exemplar este considerat ca aparținând de familia în cauză). Datorită caracterului continuu al valorilor prezise, sunt generate în primul rând, pentru fiecare clasă, metricile specifice regresie. Ulterior, clasele sunt binarizate conform cu limita stabilită pentru a calcula metrici specifice clasificărilor: matrice de confuzie, acuratețe, predicție, reamintire și coeficientul de corelație al lui Matthews.

Modelele antrenate și evaluate pot fi folosite ulterior în producție, pentru **prezicerea unor rezultate** pentru fișiere ce nu au mai fost procesate până în acel moment de model. Cum timpul de extragere a atributelor poate fi unul ridicat din cauza tipului de analiză efectuat, predicția a fost divizată în doua etape. Prima este de creare a unui tichet prin specificarea fișierului analizat, a modelului și a configurației folosite. Urmează o etapă de verificare, care poate returna fie un rezultat nul, dacă predicția pentru fișierul în cauză nu este finalizată, fie rezultatul predicției în cazul contrar.

Acuratețea modelelor poate fi crescută folosind mecanismul de publicare de rezultate ale unor analize detaliate în combinație cu cel de **reantrenare** a modelelor. Acesta din urmă poate fi atât instant, la comanda administratorului, cât și periodic, prin stabilirea unui interval la care reantrenarea să aibă loc.

Alte operațiuni disponibile sunt cele specifice gestionării de modele: listare, actualizarea unor parametrii folosiți în clasificarea (în fișier curat, suspect sau maliționos) unui fișier în funcție de maliția prezisă pentru el și ștergere.

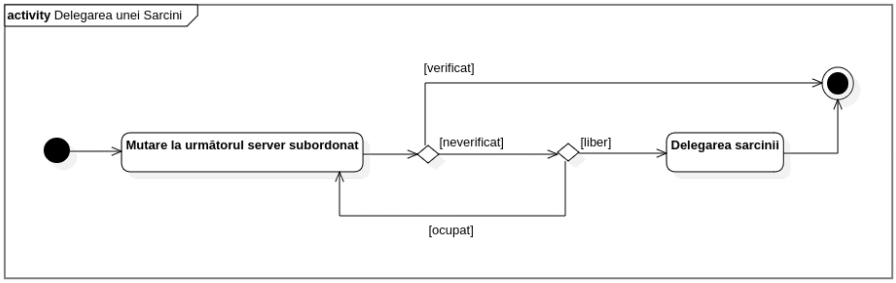
3.2.2.5 Modul pentru Gestionarea Serverelor Subordonate

După cum am descris anterior, platforma are o arhitectura cu lider și subordonați, în care un server central deleagă operațiuni intensive computațional unor servere capabile din punct de vedere al resurselor. Toate problemele care intervin în procesul de gestionare a acestora sunt rezolvate în cadrul unui modul, folosit numai de către serverul lider.

În primul rand, el se ocupa de **gestionarea conexiunilor** dintre serverul lider și cele subordonate. Cum ultimele au servicii RPC, ce ascultă pe un port anume, modulul oferă posibilitatea administratorului de a se conecta la un server anume, condiția fiind ca el să cunoască perechea de adresă - port. Alte opțiuni sunt de a scana o rețea întreagă (cu scopul de a se conecta la toate serverele identificate) și de listare a conexiunilor active.

Pe de alta parte, modulul joacă **rol de dispecer**, folosind o listă de conexiuni și un algoritm de planificare de tip Round Robin. Atunci când o sarcină trebuie delegată, el verifică următorul server din listă, după cel căruia i-a fost atribuită o sarcină la ultima execuție. În cazul în care acesta este liber (nu execută în acel moment o altă sarcină), sarcina îi este atribuită lui. Altfel, cursorul din listă este avansat până la găsirea unui server liber sau la finalizarea interogării tuturor serverelor. În ultimul caz, în care toate sunt ocupate, planificarea se încheie fără succes, iar sarcina este abandonată.

Abordarea descrisă anterior are avantajul de a evita apariția unor serverele care nu vor executa niciodată o sarcină. De exemplu, dacă lista era parcursă de la început spre final, liniar și fără ciclicitate, iar sarcinile erau suficient decalate între ele pentru a le asigura execuția completă, primele servere din listă erau solicitate constant, în timp ce ultimele rămâneau inactive pentru totdeauna.



Imagine 3.6: Diagrama de activitate pentru delegarea unei sarcini

De menționat este că **sarcinile** pot fi de două tipuri: sincrone, în care serverul lider așteaptă finalizarea execuției cu scopul de a-i procesa rezultatul (de exemplu, afișare pe ecran), și asincrone, în care serverul lider continuă să preia comenzi de la administrator și serverul delegat să își execute sarcina.

3.3 Testare

Am efectuat o **testare semi-automată a platformei**, asemănătoare conceptual cu cea de **integrare ascendentă**. Dezvoltarea a plecat de la modulele individuale, care nu depindeau de altele, și a continuat cu cele ce aveau legături cu modulele deja implementate. Pentru fiecare etapă a dezvoltării, am realizat *script*-uri pentru testarea funcționalității modulului implementat și, implicit, a eventualelor legături cu alte. Rezultatele acestora au fost verificate manual, asigurând corectitudinea lor.

După implementarea modulelor și a componentelor platformei care pot comunica direct cu utilizatorii, le-am asigurat o testare manuală prin:

- Rularea în linie de comandă a comenzilor interfeței administratorului;
- Apelarea din linie de comandă, cu ajutorul clientului HTTPie[14], a rutelor API-ului; și
- Verificarea în *browser* a funcționalităților disponibile în interfața grafică web.

Am ținut cont de posibilele înlănțuiri ce pot apărea între comenzi, apeluri de rute ale API-ului sau funcționalități din interfața grafică web întrucât acestea sunt capabile, de regulă, de a duce un program într-o stare impredictibilă, în care funcționarea nu mai este corectă. În plus, la fiecare modificare a codului sursă în urma descoperirii unei funcționări necorespunzătoare, am efectuat **testare de regresie**, asigurându-ne că platforma are un comportament complet determinist.

Menționăm faptul că API-ul a fost testat și cu ocazia integrării soluției software în cadrul unui alt program, numit **SDACIA**. Acesta a fost dezvoltat în limbajul de programare C++, pentru proiectul de cercetare "*Sistem de detecție a atacurilor cibernetice asupra sistemelor informatice folosind tehnici de inteligență artificială*", realizat în Academia Tehnică Militară "*Ferdinand I*" București. Funcționalitatea programului este de a scanare orice fișier dintr-un folder monitorizat, cu ajutorul unei serii de motoare antivirus printre care se numără și platforma prezentată în această lucrare.

[14] https://httpie.io

Capitolul 4:
Evaluarea Performanțelor

Odată finalizate proiectarea, implementarea și testarea, am continuat cu evaluarea performanțelor platformei. Cum principalul scop al acestei lucrări este de a detalia o posibilă abordare în ceea ce privește implementarea unei soluții software pentru analiza automată de programe malițioase folosind tehnici de inteligență artificială, nu am pus accentul pe optimizarea la maxim, din punct de vedere al timpului de execuție, a diferitelor operațiuni ce pot fi deprinse în cadrul platformei. În schimb, am urmărit o implementare corectă, cu rezultate cât mai bune în ceea ce privește procesul de analiză.

4.1 Seturi de Date Folosite

Pentru efectuarea evaluării performanțelor, am creat în primă instanță, cu ajutorul interfeței în linie de comandă destinate administratorilor platformei, **5 seturi de date**. În funcție de tipul de model care le va folosi, parametrii configurabili ai acestora variază conform tabelului 4.1.

Seturile de date cu număr mic de fișiere incluse sunt destinate modelelor ce au incluși extractori intensivi computațional (`StaticOpcodes` și `StaticAPIs`), pentru a asigura un timp de antrenare (dintre care extragerea de atribute acoperă cea mai semnificativă parte) relativ mic.

Majoritatea seturilor de date nu limitează fișierele incluse la anumite familii de programe malițioase. Excepție face ultimul, `PE_BACKDOOR_VS_ENCRYPTER`, care este folosit pentru diferențierea dintre programele de tip *Backdoor* și cele de tip *Encrypter*.

ID	Formatul fișierelor	Numărul de exemplare incluse	Raportul dintre fișiere benigne și malițioase	Maliție minimă	Familii de programe malițioase
SMALL_OLE	OLE	200	0.5	0.7	Toate
LARGE_OLE	OLE	700	0.1	0.7	Toate
SMALL_PE	PE	200	0.5	0.9	Toate
LARGE_PE	PE	1000	0.5	0.9	Toate
PE_BACKDOOR_VS_ENCRYPTER	PE	200	0.0	0.8	*Backdoor*, *Encrypter*

Tabel 4.1: Configurația Seturilor de Date Create

4.2 Modele Antrenate

În continuare, după finalizarea creării seturilor de date, am antrenat cu ajutorul aceleiași interfețe în linie de comandă **6 modele de învățare automată**. Din punct de vedere al obiectivelor lor, 4 dintre ele se ocupă cu regresia maliției (2 pentru formatul PE și 2 pentru OLE) și 2 pentru clasificarea în familii de programe malițioase, mai concret specializate în diferențierea dintre exemplarele de tip *Backdoor* și cele de tip *Encrypter*. Configurațiile lor sunt prezentate în tabelele 4.2 - 4.7.

Descriere	Regresia maliției pentru fișiere OLE, pe baza unui set de date de dimensiuni mici
ID-ul setului de date	`SMALL_OLE`
Extractori	`GENERAL_OLE_DETAILS`, `OLE_MACROS`
Preprocesoare	`IDENTITY`, `K_BINS_DISCRETIZER`, `N_GRAMS`

Tabel 4.2: Configurația Modelului `SMALL_OLE`

Descriere	Regresia maliției pentru fișiere OLE, pe baza unui set de date neechilibrat, de dimensiuni mari
ID-ul setului de date	`LARGE_OLE`
Extractori	`GENERAL_OLE_DETAILS`, `OLE_MACROS`
Preprocesoare	`IDENTITY`, `K_BINS_DISCRETIZER`, `N_GRAMS`

Tabel 4.3: Configurația Modelului `LARGE_OLE`

Descriere	Regresia maliției pentru fișiere PE, cu extractori ce asigură o procesare rapidă a fișierelor
ID-ul setului de date	`LARGE_PE`
Extractori	`STATIC_STRINGS`, `STATIC_PE_CHARACTERISTICS`
Preprocesoare	`IDENTITY`, `COUNTER`, `K_BINS_DISCRETIZER`, `N_GRAMS`

Tabel 4.4: Configurația Modelului `PE_STATIC_FAST_MALICE`

Descriere	Regresia maliției pentru fișiere PE, cu toți extractorii statici folosiți
ID-ul setului de date	SMALL_PE
Extractori	STATIC_STRINGS, STATIC_PE_CHARACTERISTICS, STATIC_OPCODES, STATIC_APIS
Preprocesoare	IDENTITY, COUNTER, K_BINS_DISCRETIZER, N_GRAMS, GROUP_COUNTER

Tabel 4.5: Configurația Modelului PE_STATIC_FULL_MALICE

Descriere	Clasificarea exemplarelor în familii de programe malițioase PE, cu extractori ce asigură o procesare rapidă a fișierelor
ID-ul setului de date	PE_BACKDOOR_VS_ENCRYPTER
Extractori	STATIC_STRINGS, STATIC_PE_CHARACTERISTICS
Preprocesoare	IDENTITY, COUNTER, K_BINS_DISCRETIZER, N_GRAMS

Tabel 4.6: Configurația Modelului PE_STATIC_FAST_CLASSIFICATION

Descriere	Clasificarea exemplarelor în familii de programe malițioase PE, cu toți extractorii statici folosiți
ID-ul setului de date	PE_BACKDOOR_VS_ENCRYPTER
Extractori	STATIC_STRINGS, STATIC_PE_CHARACTERISTICS, STATIC_OPCODES, STATIC_APIS
Preprocesoare	IDENTITY, COUNTER, K_BINS_DISCRETIZER, N_GRAMS, GROUP_COUNTER

Tabel 4.7: Configurația Modelului PE_STATIC_FULL_CLASSIFICATION

Diferențele dintre cele două modele pe fiecare pereche menționată de obiectiv - format constau în dimensiunea setului de date utilizat și/sau în extractorii și preprocesoarele folosite. Am dorit să observăm prin aceste variații diferențele ce apar în metricile considerate, de evaluare a performanțelor, prin prisma noilor informații deduse de către algoritmul de învățare automată, din exemplarele și/sau atributele suplimentare.

4.3 Evaluarea Performanțelor Modelelor

De menționat este faptul că înainte de antrenarea modelelor de mai sus, am efectuat o comparație între performanțele unui model asemănător celui cu codul de identificare PE_STATIC_FAST_MALICE, în care am variat prima oară tipul de algoritm de reducere a dimensionalității. Conform cu tabelul 4.8, am observat faptul că **algoritmul PCA** oferă cele mai bune rezultate considerând cele mai

relevante două metrici de evaluare, și anume rădăcina erorii medie pătratice și scorul R^2.

Am procedat în aceeași manieră și la alegerea algoritmului de învățare automată, folosind aceeași configurație, dar în care am fixat PCA ca algoritm de reducere a dimensionalității și l-am variat pe cel de învățare automată. Astfel, am stabilit pe baza informațiilor din tabelul 4.9 faptul că toate modelele ale căror performanțe vor fi expuse în acest capitol trebuie să folosească PCA, cu variația minimă înglobată în caracteristicile selectate de 0.999999, și ***Random Forest***, cu un raport dintre numărul de exemplare folosite pentru antrenare și a dimensiunii întregului set de date de 0.8.

Am evaluat automat cele 6 modele de inteligență artificială, prin intermediul modulului specific de gestionare. Am listat în tabelele 4.10, 4.11 și 4.12 rezultatele obținute. Întrucât în cazul modelelor de clasificare a fost nevoie să efectuăm o dublă relativizare, am ales cele două atribute pe care este specializat setul de date `PE_BACKDOOR_VS_ENCRYPTER` și pragul de binarizare de 0.1.

Algoritmi de reducere folosit	Eroare maximă	Eroare medie absolută	Rădăcina erorii medie pătratice	Scorul R^2
PCA	0.442	0.0991	0.1339	0.9108
NMF	0.521	0.0835	0.1563	0.8824
ICA	0.9126	0.1485	0.2809	0.6138

Tabel 4.8: Evaluarea Modelelor Antrenate cu Diferite Tipuri de Algoritmi de Reducere a Dimensionalității

Algoritm de învățare automată folosit	Eroare maximă	Eroare medie absolută	Rădăcina erorii medie pătratice	Scorul R^2
Regresie pe bază de arbori de decizie	0.9172	0.0757	0.2559	0.6812
Mașini cu vectori de suport	4.2591	0.3543	0.8243	-2.2945
Păduri aleatoare	0.5846	0.0835	0.1558	0.883

Tabel 4.9: Evaluarea Modelelor Antrenate cu Diferite Tipuri de Algoritmi de Învățare Automată

ID	Eroare maximă	Eroare medie absolută	Rădăcina erorii medie pătratice	Scorul R^2
SMALL_OLE	0.043	0.0104	0.014	0.9985
LARGE_OLE	0.0384	0.0068	0.0097	0.9972
PE_STATIC_FAST_MALICE	0.7169	0.0303	0.0743	0.974
PE_STATIC_FULL_MALICE	0.5137	0.1155	0.1702	0.8595

Tabel 4.10: Evaluarea Modelelor de Regresie a Maliției

Pentru familia *Encrypter*	
Eroare maximă	0.0802
Eroare medie absolută	0.0082
Rădăcina erorii medie pătratice	0.0169
Scorul R^2	0.5625
Acuratețe	0.975
Precizie	0
Reamintire	0
Coeficientul lui Matthews	0
Pentru familia *Backdoor*	
Eroare maximă	0.0458
Eroare medie absolută	0.0034
Rădăcina erorii medie pătratice	0.0103
Scorul R^2	0.89
Acuratețe	1
Precizie	0
Reamintire	0
Coeficientul lui Matthews	0

Tabel 4.11: Evaluarea Modelului `PE_STATIC_FAST_CLASSIFICATION`

Pentru familia *Encrypter*	
Eroare maximă	0.0721
Eroare medie absolută	0.0061
Rădăcina erorii medie pătratice	0.0139
Scorul R^2	0.6108
Acuratețe	0.975
Precizie	0
Reamintire	0
Coeficientul lui Matthews	0
Pentru familia *Backdoor*	
Eroare maximă	0.0364
Eroare medie absolută	0.0019
Rădăcina erorii medie pătratice	0.0068
Scorul R^2	0.9682
Acuratețe	1
Precizie	1
Reamintire	1
Coeficientul lui Matthews	1

Tabel 4.12: Evaluarea Modelului `PE_STATIC_FULL_CLASSIFICATION`

4.3.1 Regresia Maliției

Pentru **modelele de regresie a maliției** pentru formatul O LE, am observat din valorile erorilor maxime faptul că nu există discrepanțe în ceea ce privește predicția maliției. Nu apar scenarii cu eroare maximă mare (de exemplu, mai mare de 0.5), în care modelul să fi prezis despre un fișier malițios (cu maliție apropiată de 1) că este benign (cu maliție apropiată de 0) sau viceversa. Modelul `LARGE_OLE`, ce a dispus de un set de date de date mai mare, dar neechilibrat corespunzător, prezintă valori mai mici ale erorilor și una aproape egală (cu diferență de numai 0.13%) pentru scorul R^2.

Pentru formatul PE, erorile maxime indică apariția tipului de discrepanțe, menționat în cazul modelelor antrenate pentru formatul OLE. Eroarea medie absolută și rădăcina erorii medie pătratice au valori scăzute, iar scorurile R^2 sunt apropiate de 1. Pe de altă parte, prin comparație între modelele `PE_STATIC_FAST_MALICE` și `PE_STATIC_FULL_MALICE`, putem observa prin scăderea performanțelor faptul că un număr mai mare de atribute oferite celui de-al doilea model (și anume statisticile referitoare la grupurile de coduri de operație și de apeluri de sistem, rezultate în urma analizei cu Ghidra) nu compensează diferența în exemplarele analizate (de 800 de fișiere, dintre care 400 benigne și 400 malițioase).

4.3.2 Clasificare în Familii de Programe Malițioase

Modelele de clasificare prezintă aceleași valori ale metricilor de evaluare a performanței, specifice clasificării și aplicate după binarizarea rezultatului continuu, în ciuda numărului mai mare de atribute procesate și a folosirii aceluiași set de date. Pe de altă parte, observăm mici îmbunătățiri în privința metricilor specifice regresie.

Valorile întregi 0 și 1 sunt cauzate de faptul că cele două clase, *Backdoor* și *Encrypter*, sunt reprezentate necorespunzător în setul de date. Toate exemplarele selectate, deși sunt cele mai reprezentative pentru clasele menționate, au valori mici ale etichetelor, ceea ce provoacă o clasificare necorespunzătoare după binarizare.

4.4 Evaluarea Performanțelor Analizei de Similaritate

Pentru **evaluarea ultimului obiectiv** pe care îl poate servi un model antrenat, am efectuat o serie de pași care să ne indice faptul că acesta funcționează corespunzător. Din cauza dificultății de cuantizare a rezultatelor, ce nu apărea și în cazul obiectivelor evaluate anterior, metodologia a implicat o abordare semi-automată după cum urmează.

În primul rând, am creat manual un set de date cu 9 exemplare, dintre care:

- ***8 cu entropie mai mare de*** 7.9: Acest lucru indică împachetarea executabilului[1] și implică un număr mic de șiruri de caractere printabile (în medie, 46.5 de astfel de șiruri cu lungimea mai mare de 9 caractere).
- ***Unul cu entropie*** 5.784: Entropia mică indică lipsa împachetării, de unde și numărul mare de șiruri de caractere printabile cu lungime mai mare de 9 caractere, de 85.

Numărul mic de exemplare nu a fost ales pentru a ajuta algoritmul de învățare să obțină performanțe ridicate, ci numai pentru a observa dacă va reuși să diferențiere cele două tipuri de fișiere incluse pe baza atributelor lor.

Am antrenat apoi un model de regresie a maliției (ce avea inclus automat și obiectivul de analiză de similaritate) cu un singur extractor, `StaticStrings`, și cu preprocesorul `NGrams` atașat. Configurația reducerii dimensionalității și a antrenării a fost identică celei ale modelelor analizate anterior.

Am scanat apoi un fișier din setul de date al platformei, cu entropia 6.4 și 29 de șiruri de caractere printabile cu lungime mai mare de 9. Rezultatul returnat a indicat cea mai mare similaritate pentru fișierul cu entropia cea mai mică din setul de date, de unde reiese reușita modelului de a deduce asemănări între valorile atributelor.

[1] Robert Lyda and James Hamrock. "Using Entropy Analysis to Find Encrypted and Packed Malware". In: *IEEE Security Privacy* 5 (2007)

Capitolul 5:
Concluzii și Dezvoltare Ulterioară

Am demonstrat în această lucrare posibilitatea folosirii tehnicilor de inteligență artificială în analiza automată de programe. În cadrul unei soluții software, am îmbinate tehnicile specifice analizei de fișiere, ce constau în acest context în extragerea automată de atribute relevante din fișiere, cu cele de inginerie a datelor, și anume lucrul cu seturile de date, preprocesarea atributelor extrase din exemplare, reducerea dimensionalității, antrenarea și evaluarea unor algoritmi de învățare automată.

5.1 Situația Curentă

Implementarea curentă a soluției software **respectă obiectivele** stabilite în capitolul 1.4 și acoperă o proporție foarte mare din fiecare dintre cerințele funcționale și nefuncționale stabilite în capitolul 3.1. Acest ultim aspect va fi detaliat în secțiunea specifică dezvoltării ulterioare, de unde vor reieși sincopele existente.

Vom lista în continuare realizările obținute în această lucrare:

- Crearea unui **set de date etichetate**, cu programe malițioase, ce a fost postat *online* pentru a permite utilizarea lui liberă și eventuale îmbunătățiri datorate comunității;
- Crearea de **metode automate de extragere a unor atribute** relevante din fișiere cu format PE și OLE;
- Crearea unui **proces automatizat de inginerie a datelor**, cu pași de preprocesare, reducere a dimensionalității, antrenare și evaluare de modele de învățare automată;
- Integrarea celor menționate în cadrul unei platforme, care oferă următoarele **beneficii**:
  - Pentru **specialiștii în securitate cibernetică**, scanarea de fișiere malițioase pentru detecția maliției, clasificare și analiză de similaritate, plus posibilitatea de publicare a rezultatelor analizelor amănun-

țite, ale căror detalii sunt luate în considerare în cadrul reantrenărilor manuale sau programate a modelelor;
  - Pentru **utilizatorii normali**, scanarea de fișiere executabile și Office printr-o interfață facilă.
  - Pentru **administratorul de platformă**, oferirea unei interfețe puternic înzestrată cu funcționalități, ce permite gestionarea holistică a proceselor din platformă;
  - Pentru **administratorii de sistem**, **gestionarea ușoară** a platformei într-o infrastructură modernă și **optimizarea folosirii resurselor**, prin paralelizare și distribuirea sarcinilor cu ajutorul infrastructurii cu lider și subordonați;
  - Pentru **organizație**, **posibilitate de integrare cu alte soluții software**, datorită API-ului, și **costuri minime**, datorită caracterului de sursă deschisă; și
- Integrarea soluției software dezvoltate în cadrul unui **proiect de cercetare** al Academiei Tehnice Militare "*Ferdinand I*" București.

Din punct de vedere al modelelor ce au fost antrenate în cadrul platformei, subliniem trei rezultate bune, exprimate în rădăcina erorii medie pătratice:

- Aproximativ 0.01 pentru **modelul de regresie a maliției pentru fișiere OLE**, cu antrenare pe un set de date mare și cu toate atributele (relativ la tehnicile implementate) extrase;
- 0.0743 pentru **modelul de regresie a maliției pentru fișiere PE**, cu analiză a atributelor extrase static, fără ajutorul dezasambloarelor; și
- 0.0139 și 0.0068 pentru **modelul de clasificare a programelor malițioase de tip *encrypter* și *backdoor***, folosind toate atributele statice (relativ la tehnicile implementate) extrase.

5.2 Dezvoltare Ulterioară

Această secțiune cuprinde aspectele pe care le vom avea în considerare în **dezvoltarea ulterioară** a lucrării și a soluției software implementate.

5.2.1 Îmbunătățiri

- **Echilibrarea setului de date**, prin asigurarea unui raport egal între fișierele malițioase și cele benigne și a unui număr egal de fișiere per familie de programe malițioase
- **Extinderea funcționalității extractorilor dinamici** bazați pe Qiling, care suportă în momentul de față un număr redus de fișiere din cauza apelurilor de sistem neimplementate

- **Refactorizarea codului**, inclusiv prin simplificarea anumitor clase și metode
- **Continuarea optimizărilor**, prin rularea dezasamblorului Ghidra pe mai multe fire de execuție și prin păstrarea în memorie a seturilor de date pentru a crește viteza lor de procesare

5.2.2 Funcționalități Noi

- Crearea unei **fabrici de seturi de date**, posibil cu ajutorul unei integrări cu VirusTotal Intelligence[1]
- Crearea unor **extractori noi cu tehnici de analiză dinamică** (de exemplu, pentru lucru cu sistemul de fișiere, activitate la nivel de rețea și de regiștrii)
- Studierea **scalării soluției la organizațiile mari**

[1] https://www.virustotal.com/gui/intelligence-overview

Bibliografie

Cărți

[1] Andriy Burkov. *The Hundred-Page Machine Learning Book*. 2019.

[2] Michael Sikorski. *Practical Malware Analysis: The hands-on guide to dissecting malicious software*. San Francisco, CA: No Starch Press, 2012.

Articole Științifice

[3] Daniele Ucci, Leonardo Aniello, and Roberto Baldoni. "Survey of machine learning techniques for malware analysis". In: *Comput. Secur.* 81 (2019).