UNIVERSITATEA BABEŞ-BOLYAI CLUJ-NAPOCA

FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ

REZUMAT TEZĂ DE DOCTORAT

Metode de instruire automată în analiză malware

Attila Mester

Coordonator științific: prof. dr. Anca Andreica

2025

Cuvinte cheie: analiza statică a programelor malware, inginerie inversă, IDA, Radare2, atribuirea APT

Abstract

Amenințările cibernetice reprezintă o preocupare în continuă creștere atât pentru organizații, cât și pentru persoane. Conform statisticilor recente, la nivel global, în fiecare secundă apar aproximativ trei noi fișiere potențial periculoase. Această cantitate de date este imposibil de prelucrat manual de către companiile de securitate cibernetică, prin urmare, sunt utilizate instrumente automate pentru a ajuta la identificarea și clasificarea acestor amenințări. Raportul de informații privind amenințările al unui instrument de securitate conține etichete precum verdictul (malițios sau nu), etichetele de detectare și, de asemenea, *attribution*, privind familia fișierului sau grupul de atacatori care l-au creat.

Această teză prezintă o comparație între două instrumente de dezasamblare, IDA și Radare2, concentrându-se pe generarea grafului static de apeluri. Apoi, prezentăm noi metode de extragere a caracteristicilor cu scopul de a atribui unui fișier malware o anumită familie. Sunt explorate două direcții principale, gruparea și clasificarea programelor malware, ambele valorificând graful static de apeluri.

În primul rând, propunem o nouă metodă de extragere a semnăturii din graficul static de apeluri, care este utilizată pentru a grupa fișierele malițioase într-un grafic bazat pe amprente comune. După gruparea fișierelor cu ajutorul algoritmilor de detectare a comunităților, arătăm că grupurile rezultate sunt foarte omogene în ceea ce privește familiile malware.

În al doilea rând, sunt prezentate două noi metode de clasificare: una dintre acestea se bazează pe rețele neuronale convoluționale pe grafuri, antrenate pe grafuri de apeluri statice, iar cealaltă oferă o metodă de codificare pentru a converti graful de apeluri static în imagini RGB, transformând astfel problema clasificării programelor malware într-una de clasificare a imaginilor.

De asemenea, publicăm un nou set de date malware pe Kaggle, împreună cu diverse informații de analiză legat de MalImg și BODMAS.

Conținut

Capitolul 1

Introducere

Digitalizarea a crescut numărul de amenințări online cu care ne confruntăm în fiecare zi. Deși majoritatea atacurilor cibernetice vizează marile corporații, persoanele fizice sunt, de asemenea, expuse riscului. Termenul comun pentru a descrie aceste amenințări este malware, care înseamnă software malițios, adică un tip de program sau fișier informatic care urmărește să efectueze acțiuni dăunătoare pe un computer. Conform AV-TEST[1], numărul de amenințări noi care apar zilnic este de aproximativ 3/sec, sau $300k$/zi.

1.1 Obiective

Scopul nostru este de a dezvolta noi metode pentru analiza, gruparea și clasificarea programelor malware pe baza grafurilor statice de apelare a fișierelor malware. Ne concentrăm pe prelucrarea rapidă și scalabilă a acestor entități, fie prin extragerea semnăturilor sensibile la localitate din graf, fie prin introducerea grafului în rețele neuronale care pot învăța modele utile ale topologiei sale. În plus, ne propunem să îmbogățim comunitatea prin publicarea seturilor noastre de date, a codului sursă al metodelor noastre și a unei analize ample a seturilor de date deja existente.

1.2 Contribuții

Contribuțiile acestei teze sunt următoarele (rezumate pe pagina noastră GitHub Page[2]).

[1] `https://www.av-test.com`

[2] `https://attilamester.github.io/call-graph`

(i) Sunt prezentate trei metode noi în domeniul clasificării programelor malware. Introducem o nouă metodă de extragere a semnăturilor bazată pe graful static de apeluri al unui executabil. Aceste semnături sunt apoi utilizate pentru a grupa fișierele prin aplicarea algoritmului de detectare a comunității Louvain pe graficul malware. Demonstrăm că metoda noastră poate grupa fișierele malițioase în grupuri omogene, în funcție de eticheta de familie reală, prin compararea acestora cu semnăturile brevetate utilizate la nivel industrial.În plus, prezentăm două abordări noi pentru clasificarea codurilor malware în familii. Prima metodă utilizează rețele neuronale convoluționale grafice pentru a învăța modelele grafurilor de apeluri statice ale fișierelor malițioase. A doua metodă transformă graficul static de apeluri într-o imagine și utilizează arhitecturi CNN bine cunoscute pentru a clasifica fișierele malware.

(ii) Publicăm un proiect GitHub, *malflow*[3], care conține codul sursă al algoritmului nostru static de generare a grafului de apeluri. Proiectul conține, de asemenea, codul sursă al algoritmilor de generare a imaginilor propuși de noi și schemele de codificare a instrucțiunilor.

(iii) Analizăm seturile de date malware publice, considerate repere în domeniu, și anume BODMAS și MalImg, și introducem, de asemenea, un nou set de date, Internal Bitdefender Dataset (IBD), în cadrul proiectului public GitHub *malflow*. Publicăm următoarele:

- obiecte dezasamblate Radare2, care conțin graficul complet al apelurilor statice, cu nume de funcții, apeluri și, de asemenea, informații complete despre instrucțiuni, cum ar fi mnemonica, prefixele, operanzii și adresa instrucțiunii;
- diverse statistici privind instrucțiunile extrase din aceste fișiere malițioase, cum ar fi distribuția acestora și ordinea de apariție în funcție de execuția simulată a fișierului;
- informații packer, obținute prin utilizarea instrumentului DIE;
- trei tipuri de imagini RGB pentru fiecare sample, pe baza diferitelor codificări care atribuie un pixel unei instrucțiuni. Aceste imagini sunt utilizate

[3] `https://github.com/attilamester/malflow`

în a doua noastră abordare de clasificare, în care transformăm graficul static al apelurilor într-o imagine și utilizăm arhitecturi CNN pentru a clasifica malware.

(iv) Descoperim o corelație între familiile APT și packer-ul utilizat pentru a ofusca fișierul executabil. Aceste observații sunt publicate pentru trei seturi de date diferite care se întind pe aproape 15 ani, MalImg, BODMAS și IBD, și sunt publicate în contextul proiectului *malflow*.

1.3 Lista de publicații

Standardele de evaluare utilizate sunt cele valabile la 1 octombrie 2018. Lista conferințelor, precum și categorisirea acestora, se bazează pe clasamentul internațional al conferințelor *CORE*[4]. Lista revistelor este cea utilizată de *UEFISCDI*[5] pentru premierea articolelor publicate în reviste științifice internaționale.

- **A. Mester**, A. Pop, B. Mursa, H. Grebla, L. Diosan, C. Chira (2021). *Network Analysis Based on Important Node Selection and Community Detection*. Mathematics, 9.18, 2294
- **A. Mester**, Z. Bodó, P. Vinod, M. Conti (2025). *Towards a malware family classification model using static call graph instruction visualization*. 18th International Conference on Network and System Security, Network and System Security, 167–186, Springer Nature Singapore
- **A. Mester**, Z. Bodó (2021). *Validating static call graph-based malware signatures using community detection methods*. 29th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning, ESANN 2021, 429–434
- **A. Mester**, Z. Bodó (2023). *Malware classification based on graph convolutional neural networks and static call graph features*. Advances and Trends in Artificial Intelligence. Theory and Practices in Artificial Intelligence: 35th International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems, IEA/AIE 2022, 528–539, Springer
- **A. Mester** (2023). *Malware analysis and static call graph generation with Radare2*. In Studia Universitatis Babeș-Bolyai Informatica, 68.1, 5–20

[4] `https://portal.core.edu.au/conf-ranks`
[5] `https://uefiscdi.gov.ro/scientometrie-reviste`

Capitolul 2

Analiză malware

Scopul acestei teze este analiza statică a programelor malware sub formă de fișiere executabile portabile (PE). Fișierele PE pot conține informații despre amenințări cu privire la originea atacului. O astfel de informație extrem de valoroasă este denumită *attribution/atribuire* în literatura de specialitate [Steffens, 2020].

Atribuirea informațiilor APT atacurilor cibernetice este o prioritate majoră a organizațiilor de securitate cibernetică. Atribuirea cuprinde informații despre autorii atacului și familiile de programe malware desfășurate. Analiștii manuali nu pot procesa cantitatea mare de fișiere și evenimente care declanșează alarme în soluțiile de securitate cibernetică în timp real. Prin urmare, analiza automată a mostrelor de malware este esențială pentru produsele antivirus.

Există două opțiuni fundamentale privind analiza unui fișier executabil: analiza statică și analiza dinamică. Aceste metode presupun utilizarea fie a unui mediu sandbox – care este adesea costisitor și consumator de timp, fie a unui instrument de dezasamblare precum IDA, Radare2, Ghidra etc., după cum se detaliază în Capitolul 3. Ambele metode pot fi efectuate manual de către analiști sau automat, folosind scripturi automate și diverse instrumente.

Analiza statică se limitează la inspectarea conținutului fișierului pe disc, fără a-l executa. Această metodă este mai rapidă și consumă mai puține resurse decât analiza dinamică, oferind proprietăți simple ale fișierelor binare (de exemplu, dimensiunea fișierului, informații despre secțiuni, tabel de import și export) și, de asemenea, caracteristici mai complexe, cum ar fi distribuția de octeți, instrucțiunile dezasamblate și reprezentările lor mai complexe, cum ar fi graficul fluxului de control (CFG) sau graficul de apeluri. Cu toate acestea, este posibil să nu poată detecta tot comportamentul

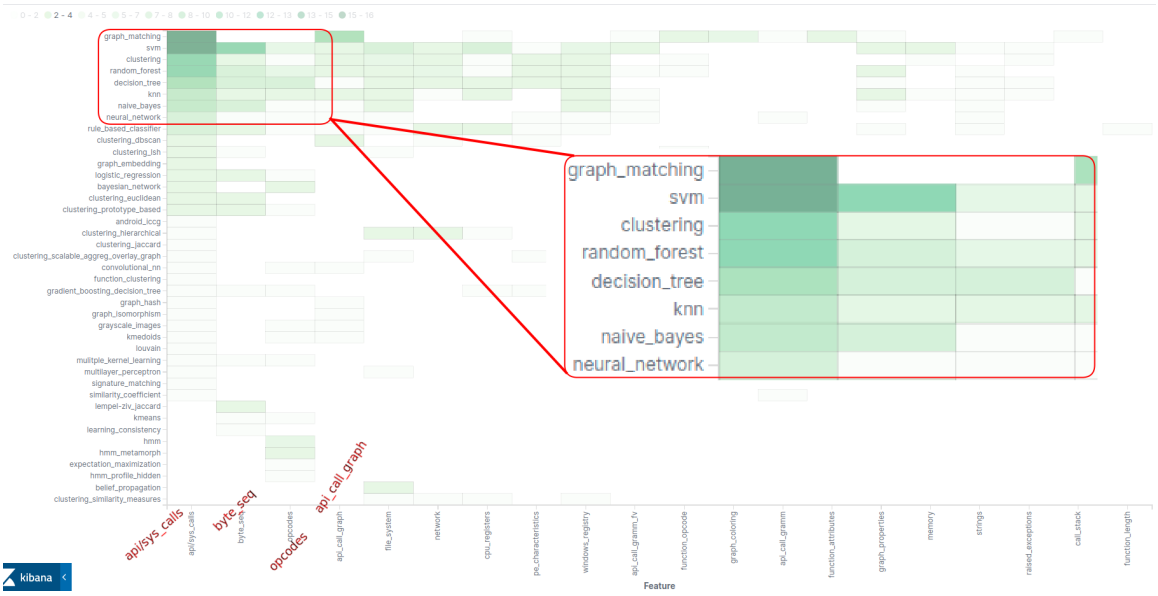


Figura 2.1: Cei mai frecvent utilizați algoritmi sunt aplicați pe caracteristici selectate din apelurile API/sistem. Acest grafic agregă rezultatele unui studiu [Ucci et al., 2019] care clasifică aproximativ o sută de lucrări de cercetare în domeniul analizei programelor malware în funcție de caracteristicile extrase și algoritmii aplicați.

malițios al fișierului, în special dacă malware-ul este criptat sau ofuscat.

Analiza dinamică necesită o tehnologie sandbox, care oferă un mediu izolat în care malware-ul să fie executat, simulând un scenariu real de infectare. Sandbox-ul va urmări evenimentele declanșate de acest fișier malware, cum ar fi evenimente ale sistemului de fișiere, apeluri API, modificări ale registrului, activitate în rețea, operațiuni pe disc. Acest tip de analiză are avantajul de a oferi evenimente reale legate de activitatea malware-ului, care vor forma trăsăturile sale naturale. Cu toate acestea, caracterul său practic poate fi considerat limitat din cauza timpului de analiză mai lung, care depinde de codul util malițios, care poate sau nu poate fi activat [Quertier et al., 2022].

Vizualizarea din Figura 2.1 ne arată frecvența perechilor algoritm – caracteristică în vasta prezentare generală a literaturii [Ucci et al., 2019].Acest lucru ne arată posibilele noi direcții de cercetare, dar cel mai important este faptul că cele mai populare metode implicau aplicarea unui fel de algoritm de potrivire a grafurilor, a metodei SVM sau a oricărui tip de grupare pe date extrase din apelurile API/sistem. Acest lucru a determinat direcția cercetării noastre, cu scopul de a explora posibilitățile de valorificare a caracteristicii grafului static al apelurilor dintr-un fișier executabil.

Capitolul 3

Instrumente de dezasamblare

Analiza programelor malware presupune adesea inspectarea comportamentului fișierului PE. Analiza dinamică poate oferi inspectarea comportamentului real al fișierului atunci când este executat, dar consumă mult timp și este posibil să nu fie fezabilă pentru seturi mari de date (a se vedea Capitolul 2). Atunci când se pune accentul pe analiza comportamentală, o abordare statică poate fi mai eficientă, deoarece poate oferi o perspectivă rapidă asupra comportamentului fișierului fără a fi necesară executarea acestuia, prin analizarea codului de asamblare al fișierului. În acest scop, se pot utiliza diverse instrumente de dezasamblare pentru a inversa ingineria fișierului binar și a obține codul său de asamblare.

În acest capitol, descriem două dintre cele mai populare instrumente de dezasamblare, IDA și Radare2, și le comparăm în ceea ce privește funcționalitatea și utilitatea lor în ceea ce privește generarea de grafuri statice de apeluri ale fișierelor executabile. Aceste rezultate sunt publicate în lucrarea noastră [Mester, 2023].

3.1 Prezentare generală a instrumentelor de dezasamblare

IDA este un dezasamblator interactiv[1] și unul dintre cele mai importante instrumente de dezasamblare din industrie [Nar et al., 2019, Yin et al., 2018]. Acesta are o interfață grafică bogată și oferă, de asemenea, o gamă largă de funcționalități scriptate, disponibile prin intermediul limbajului de scripting IDA. Deși utilizarea sa este discutabil de versatilă și oferă un instrument de analiză puternic pentru ingineria

[1] `https://www.hex-rays.com/products/ida`

inversă, acesta are și limitările sale. Unul dintre dezavantajele majore este faptul că IDA este un software comercial, iar funcționalitatea scriptată este disponibilă numai în versiunea plătită, IDA Pro.

Există o multitudine de instrumente de dezasamblare alternative: Binary Ninja [Priyanga et al., 2022], Hopper [Andriesse et al., 2016], Relyze [Wenzl et al., 2019], x64dbg[2], ODA[3], etc. Una dintre cele mai populare alternative este Ghidra[4], disponibilă pe Windows/Linux, dezvoltată de NSA's Research Directorate sub licență Apache (FOSS). Conform diverselor cercetări, este o alternativă principală la IDA Pro [Shaila et al., 2021, Koo et al., 2021]. Dezavantajul acestui instrument, care ne-a determinat să alegem o altă alternativă, este dificultatea de a utiliza apelul său API scriptat/generarea de grafuri.

Radare2[5] este disponibil și pe Windows/Linux (FOSS) și oferă o alternativă ușoară la Ghidra, putând în același timp să integreze compilatorul Ghidra *r2ghidra*[6]. Acesta poate fi utilizat din CLI și, de asemenea, GUI, oferit de Cutter[7]. O putere majoră a acestui instrument este legătura Python *r2pipe*[8], care oferă API-uri extinse pentru analiza statică, inclusiv inspecția graficului de apeluri. Radare2 (denumit și *r2*) are, de asemenea, o mare popularitate în domeniul tehnologiei informatice [Massarelli et al., 2019, Cunningham et al., 2019, Gibert et al., 2020, Cohen, 2019, Kilgallon et al., 2017].

3.2 Generarea grafului de apeluri cu IDA Pro 6 și Radare2

IDA Pro 6

Cu funcționalitatea sa de scripting în IDA Pro 6, folosim două API-uri pentru a genera fișiere text în format GDL. *GenCallGdl* generează structura grafului de apel (adică un nod reprezintă o funcție, o muchie marchează un apel de funcție), în timp ce *GenFuncGdl* generează diagrama fluxului de execuție (adică un nod conține o listă

[2] `https://x64dbg.com`
[3] `https://github.com/syscall7/oda`
[4] `https://ghidra-sre.org`
[5] `https://www.radare.org`
[6] `https://github.com/radareorg/r2ghidra`
[7] `https://cutter.re`
[8] `https://r2wiki.readthedocs.io`

de instrucțiuni, iar muchiile direcționate reprezintă un salt de execuție între aceste blocuri). Graficul de apel final este o versiune prelucrată a acestor două fișiere, așa cum este detaliat în lucrările noastre anterioare [Mester, 2020, Mester, 2023].

Radare2

Radare2[5] oferă o descriere clară a instalării sale pe pagina Github[9] și dispune de multă documentație și sprijin din partea comunității pe pagina Wiki[10] și în cartea electronică oficială [Radare, 2009]. După instalare, Radare2 poate fi apelat folosind comenzile *radare2* sau *r2*, specificând o cale către un fișier PE.

În acest CLI, ni se oferă o varietate de comenzi și instrumente pentru a analiza secțiuni, importuri, exporturi și multe altele – de asemenea, fiecare comandă are o interfață ajutătoare care poate fi invocată prin adăugarea lui "?" după comanda respectivă. Radare2 lucrează cu conceptul de steaguri (flags), adică un marcaj la un offset de genul "fcn." sau "sym.imp", ceea ce înseamnă că fiecărui offset considerat interesant de Radare2 îi va fi atribuit un steag corespunzător, de exemplu șiruri de caractere, funcții și multe altele. Analiza unui fișier binar poate fi inițiată prin comanda "aaa", care analizează toate steagurile din fișier. În această lucrare ne-am concentrat pe analiza grafului static de apeluri, și vom detalia comenzile care sunt legate de analiza secvențelor de apeluri și a blocurilor de funcții.

Majoritatea acestor comenzi *r2* au mai multe formate de ieșire, disponibile prin specificarea unui formator la sfârșitul comenzii – cum ar fi arta ASCII implicită, sau "j" pentru *json*, "d" pentru *dot*, "b" pentru "Braile art", sau "w" pentru un complot interactiv.

Radare2 oferă legături Python prin intermediul pachetului *r2pipe*, care permite transmiterea mai multor comenzi *r2* fără a fi nevoie să deschidem și să încărcăm fișierul de fiecare dată. Începem analiza prin apelarea comenzii "aaa". Apoi, colectăm nodurile punctului de intrare, adică blocurile funcționale, prin apelarea comenzii "ie". Comenzile *r2* pe care le folosim pentru analiza grafului de apeluri fac parte din grupul de comenzi "ag".

Structura grafului de apeluri este furnizată de comanda "agC", în care specificăm, de asemenea, formatul DOT prin indicatorului "d". Graful de referință complet (de exemplu, importurile) este oferit de comanda "agR". Pentru a obține codul de

[9] `https://github.com/radareorg/radare2`
[10] `https://r2wiki.readthedocs.io`

asamblare al fiecărui subprogram, apelăm comanda "agf" pe fiecare nod al grafului de apeluri. În mod similar cu generarea grafului de apeluri folosind IDA Pro 6, fuzionând rezultatele "GenCallGdl" și "GenFuncGdl" [Mester, 2020, Mester and Bodó, 2021, Mester and Bodó, 2022], aceeași logică se aplică și în Radare2. Trebuie să adunăm informații atât din graficul global al funcției ("agC"), cât și din graficul global de referință ("agR") și, de asemenea, trebuie să analizăm separat fiecare bloc de funcții ("agf") pentru a obține graficul final, complet, al apelurilor.

3.3 Compararea grafurilor de apeluri obținute prin IDA Pro 6 și Radare2

O diferență esențială între IDA Pro 6 și Radare2 este că în primul, a trebuit să apelăm doar 2 API-uri, în timp ce în al doilea, trebuie să apelăm o multitudine de comenzi *r2* – $O(n)$ unde n este numărul de blocuri funcționale. Revelația neașteptată este că, în ciuda tuturor acestor aspecte menționate, Radare2 scanează binarele mult mai rapid decât IDA, iar grafurile de apeluri generate de aceste două instrumente sunt aproape identice [Mester, 2023].

3.4 Concluzii

Acest capitol a prezentat compararea a două instrumente de dezasamblare, IDA Pro și Radare2, în ceea ce privește capacitățile lor de generare a grafului static de apeluri. În timp ce IDA oferă două apeluri API pentru a genera graful de apeluri, Radare2 necesită o abordare mai complexă, utilizând biblioteca sa Python, *r2pipe*. În ambele cazuri, trebuie parcurși mai mulți pași pentru a construi un graf de apeluri final, complet.

În timpul experimentelor, a fost utilizat un set de date public pentru a oferi o transparență totală a rezultatelor. Grafurile de apeluri au fost comparate din diverse perspective, atât aspecte topologice, cât și criterii la nivel de nod, și anume lista de instrucțiuni a fiecărui subprogram. Rezultatele susțin că nu există nici o schimbare semnificativă în ceea ce privește rezultatele dezasamblărilor IDA și Radare2, cu toate acestea, cel din urmă oferă o modalitate mai rapidă de analiză prin script. Aceste rezultate au fost publicate în revista Studia Universitatis Babeș-Bolyai Informatica [Mester, 2023].

Capitolul 4

Aplicarea detectării comunităților în analiza programelor malware

Acest capitol prezintă aplicarea detectării comunităților ca metodă de grupare a fișierelor malițioase în funcție de similaritatea lor [Mester and Bodó, 2021].

O comunitate înseamnă un set de noduri, cu următoarele proprietăți: (i) acestea formează un subgraf conectat; (ii) nodurile din comunitatea A au proprietăți comune (caracteristici topice și topologice); (iii) nodurile din comunitatea A au mai multe proprietăți comune cu alte noduri din A decât cu cele găsite în comunitatea B.

O analiză detaliată a diferitelor concepte și metrici aplicabile în știința rețelelor și detectarea comunităților poate fi găsită în lucrarea noastră anterioară [Mester et al., 2021]. În această lucrare, ajungem la concluzia că, printre diversii algoritmi de clusterizare ierarhică existenți, optimizările modularității etc., algoritmul Louvain [Blondel et al., 2008] este o opțiune preferabilă pentru rețelele mari, datorită complexității sale $\mathcal{O}(\ell)$.

În acest capitol, prezentăm o metodă al cărei scop este de a construi o rețea de fișiere malware, fiecare nod reprezentând un virus, iar legăturile sunt stabilite dacă există amprente digitale comune – metoda de selecție a amprentelor digitale este partea crucială, principala noastră contribuție la acest domeniu. Semnăturile sunt obținute prin analiza statică a codului, utilizând n-grame de apeluri de funcții și aplicând tehnici de hashing sensibile la localitate pentru a permite potrivirea între funcții cu liste de instrucțiuni asemănătoare.

4.1 Analiză a literaturii de specialitate

Pe baza unui studiu recent [Ucci et al., 2019], se poate concluziona că cele mai frecvent utilizate caracteristici sunt apelurile API/sistem, secvențele de octeți și graful apelurilor API. Utilizând caracteristicile grafului API, se aplică frecvent algoritmi de potrivire a grafurilor sau se construiește un vector caracteristic al grafului [Park et al., 2010, Dahl et al., 2013].

Rețelele neuronale convoluționale sunt, de asemenea, aplicate cu succes pentru a detecta secvențele de coduri de instrucțiuni malware [McLaughlin et al., 2017, Martinelli et al., 2017]. În [Oprisa et al., 2014] s-a experimentat deja gruparea programelor malware bazată pe hashing sensibil la localitate. În [Hassen and Chan, 2017], LSH este aplicat subprogramelor locale, utilizând semnăturile minhash pentru a aproxima similaritatea dintre seturile de instrucțiuni ale două subprograme.

Metoda noastră extrage amprente multiple din graful apelurilor, permițând procesarea și gruparea continuă a noilor fișiere primite, fără a fi necesară reeducarea modelului de bază. Un alt aspect cheie al abordării noastre este și validarea caracteristicilor utilizate industrial [Topan et al., 2013].

4.2 N-grame pentru gruparea fișierelor malware

Amprentele sunt extrase din graficul static al apelurilor, obținut prin dezasamblarea IDA Pro 6[1] – a se vedea Secțiunea 3.2. O amprentă digitală este un n-gram în graful de apeluri rezultat: unigram înseamnă un cuvânt de cod pentru o subrutină, în timp ce bigram reprezintă o pereche de cuvinte de cod apelant - apelat.

Pentru a măsura utilitatea semnăturilor generate, metodele de detectare a comunității sunt aplicate pe graful malware bazat pe amprente digitale comune – nodurile reprezintă malware-ul, iar o muchie indică amprente comune – Figura 4.1. Un astfel de grafic malware poate fi văzut în Figura 4.2, unde nodurile sunt colorate în funcție de comunitățile Louvain obținute de Gephi[2].

Avem în vedere următoarele cerințe: (i) graful malware ar trebui să aibe componente dense, cu cât mai puține muchii între ele – înseamnând separarea grupurilor malware; (ii) nodurile unei componente ar trebui să aibă aceeași etichetă de familie.

[1] `https://www.hex-rays.com/products/ida`
[2] `https://gephi.org`

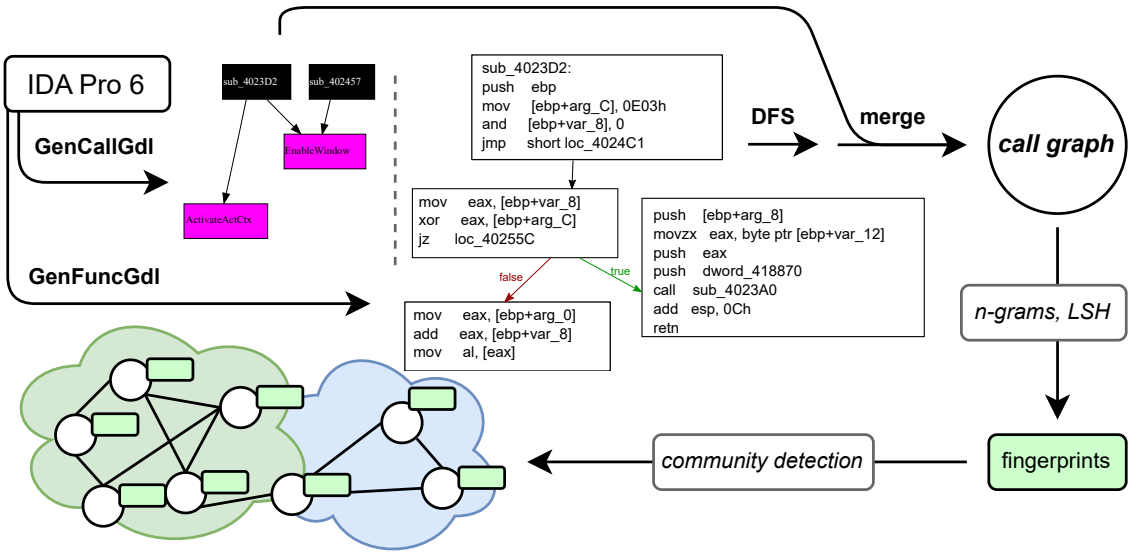


Figura 4.1: Generarea amprentelor digitale și validarea a acestora prin aplicarea metodelor de detectare a comunității pe graful malware.

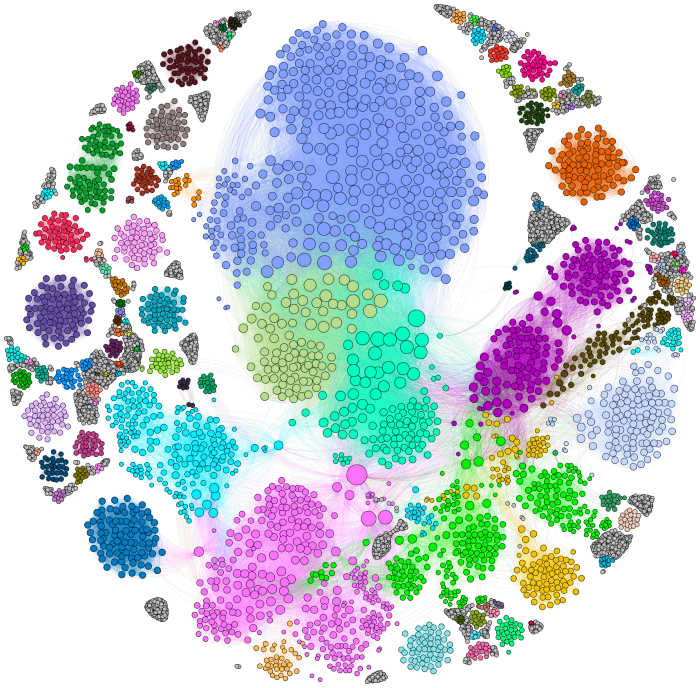


Figura 4.2: Comunitățile Louvain ale graficului malware în funcție de amprentele digitale comune (set de date intern Bitdefender de 7977 de fișiere din 254 familii, având o încredere ridicată în etichetele familiilor). Layout: Gephi's Force Atlas 2. Metoda s-a dovedit a fi bine adaptată pentru gruparea familiilor, măsurată prin scorul de omogenitate a partiției.

	Omogenitate	Completitudine	Etichetă majoritară
(a)	0.833	0.765	85%
(b)	0.632	0.704	89%

Tabela 4.1: Evaluarea numerică bazată pe familie a grafurilor malware.

4.3 Metodologia de evaluare

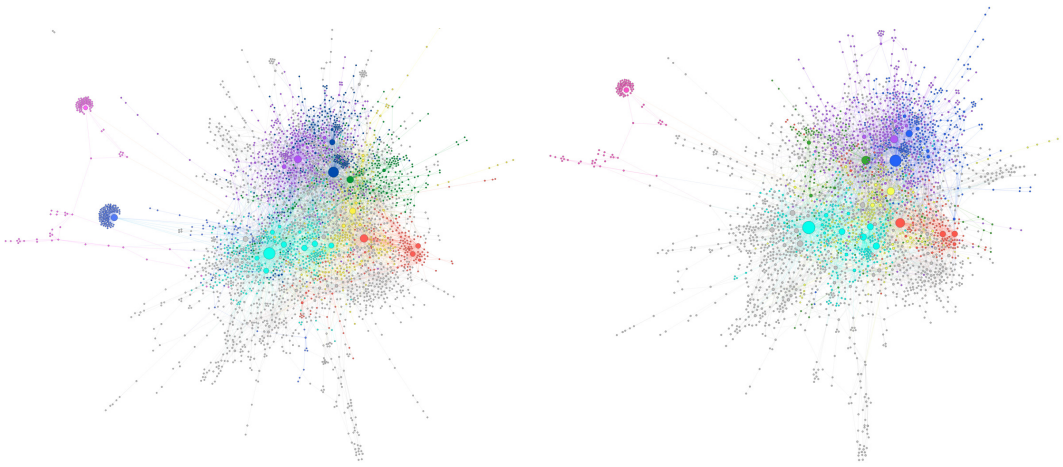
Algoritmii de detectare a comunităților cei mai de succes și utilizați frecvent sunt folosite pentru validarea abordării noastre: Clauset–Newman–Moore (CNM), algoritmul de propagare a etichetelor (LPA), Louvain și Infomap [Fortunato, 2010]. Pentru a evalua proprietățile topologice ale grafului malware, calculăm scorurile *modularity*, *coverage* și *performance* [Fortunato, 2010]. Măsurăm, de asemenea, procentul majoritar al etichetelor din cadrul fiecărui cluster, pentru a analiza distribuția familială.

4.4 Hiperparametri

În experimente, am folosit Python 3.6 (biblioteca *networkX*), IDA Pro 6, Graphviz, Gephi 0.9.2. Următoarele combinații de hiperparametri au fost experimentate: instrucțiuni de subprogram n-grame (1–3-grame, 2–3-grame sau trigrame); numărul de hiperplane aleatorii (8 sau 16); partiția de proiecție (prin proiectarea unui vector pe un hiperplan și luarea semnului proiecției pentru a genera cuvintele de cod hash, informația despre distanță se pierde – activarea acestui parametru stabilește intervale de distanță de $[0, 10]$, $(10, 100]$, $(100, 1000]$ și > 1000, și simetric cu semne negative, pentru a utiliza o partiție mai fină a spațiului); numește graficul n-grame (unigrame sau bigrame). Pe graficul final, sunt aplicate următoarele filtre: frecvența amprentelor ($[2, 80]$, $[2, 100]$, $[3, 100]$, $[10, 100]$ sau $[10, 400]$); greutatea marginilor (minimum 5, 10, 100, 200 sau 1000). Aceste parametrizări au dus la un total de 260 de execuții.

4.5 Set de date

Setul de date privat Bitdefender constă din 7977 fișiere malware din 254 familii, o familie având în medie $30,3 \pm 41,96$ fișiere. Numărul de subrutine per fișier este în medie de 1163 (mediana de 354 și maximul de 65.060) – însumând un număr total de 9.284.242 subrutine cu 650.225 secvențe mnemonice unice.



Sample s_1 Sample s_2

Figura 4.3: Compararea a două grafuri de apeluri din aceeași familie în cadrul aceleiași comunități (grupul de funcții suplimentare de pe graficul din stânga arată un pachet *winsock2*).

4.6 Concluzii

Am calculat metricile de evaluare menționate mai sus pentru diferite grafuri de malware, pe baza următoarelor amprente digitale:

(a) amprente digitale utilizate industrial [Topan et al., 2013] (4745 noduri, $55.9k$ legături)

(b) amprentele digitale descrise în această lucrare: instrucțiuni 2–3-grame, 8 hiperplane, partiție de proiecție, bigrame de graf de apel, filtrare de frecvență $[2, 100]$ și greutate minimă de 100 (4502 noduri, $114k$ legături).

Rezultatele sunt comparate în tabelul 4.1. Pe baza experimentelor, putem conclude următoarele: *(i)* amprentele digitale utilizate industrial pot grupa familii cu modularitate ridicată și scoruri procentuale ale etichetelor majoritare; *(ii)* configurația **(b)** oferă rezultate la fel de bune ca **(a)**; *(iii)* instrucțiuni n-grame, secvențele oferă o reprezentare mai bună reprezentare mai bună decât unigramele simple; nu numai codul subrutine, ci și secvențialitatea acestor subrutine poate caracteriza familiile de programe malware.

Pentru a rezuma, am prezentat o metodă nouă de extragere a amprentelor digitale din analiza statică a unui sample, prin aplicarea LSH asupra subrutinelor sale. Am arătat că metoda noastră poate fi utilizată pentru a grupa familiile de programe malware cu modularitate ridicată [Mester and Bodó, 2021].

Capitolul 5

Atribuirea programelor malware

Acest capitol prezintă problema clasificării programelor malware, și anume atribuirea familiei [Steffens, 2020]. Prezentăm două metode noi, una bazată pe rețele convoluționale grafice și cealaltă pe clasificarea imaginilor, ambele valorificând graful static de apel al fișierului malițios.

5.1 Introducere

Una dintre cele mai relevante și valoroase etichete care pot fi atribuite unui fișier malware este familia și informațiile despre autor [Ucci et al., 2019, Tahir, 2018, Steffens, 2020].

Prima noastră metodă valorifică topologia și, de asemenea, informațiile la nivel de nod ale grafului static de apeluri prin formarea unui model GCN pentru a clasifica familiile [Mester and Bodó, 2022]. Acest model a fost antrenat pe un set de date real format din mii de fișiere malițioase din 223 de familii – un număr semnificativ mai mare decât familiile utilizate în literatura de specialitate.

Cealaltă metodă convertește fișierul malițios într-o imagine și execută clasificarea familiei pe baza unei sarcini de clasificare a imaginilor. Noutatea acestei metode constă în modul în care sunt create imaginile, pe baza unei traversări specifice a instrucțiunilor din graficul static de apeluri, care simulează fluxul teoretic de execuție al programului [Mester et al., 2025].

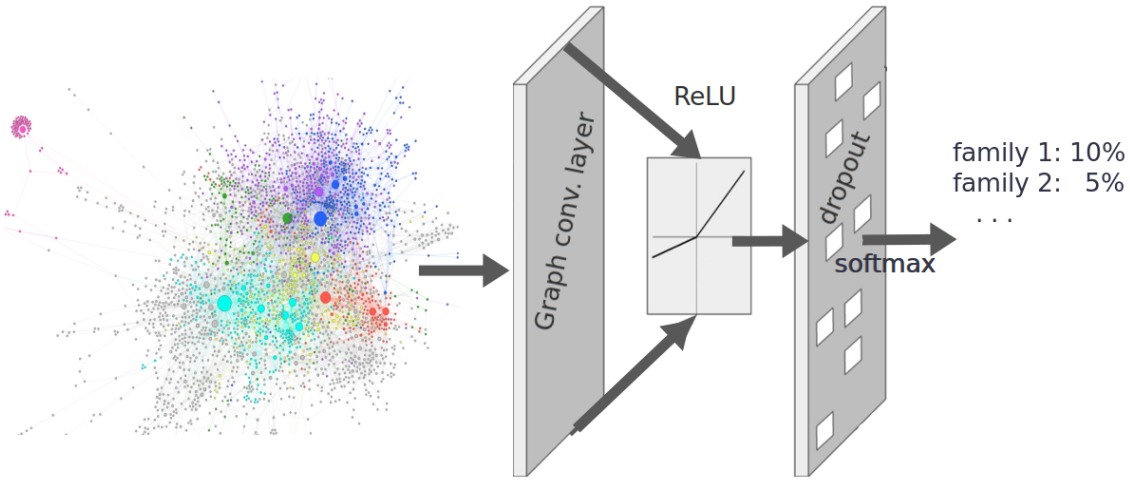


Figura 5.1: Schiță de proiect pentru atribuirea fișierelor malware utilizând GCN pe graficul static de apeluri [Mester and Bodó, 2022].

5.2 Atribuire prin GCN pe graful static de apeluri

5.2.1 Analiză a literaturii de specialitate

Apelurile API (atât dinamice, cât și statice) sunt de departe una dintre cele mai utilizate caracteristici în literatura de specialitate pentru clasificarea programelor malware [Ucci et al., 2019]. Rețelele neuronale grafice convoluționale (GCN), modelele GCN profunde (DGCNN) prezintă o popularitate crescândă în domeniul informațiilor privind amenințările cibernetice, probabil datorită noutății, precum și beneficiilor acestor abordări. În plus, putem concluziona, de asemenea, că metodele de detectare a programelor malware sunt, în general, validate folosind doar aproximativ 10 clase. Apelurile API și funcțiile obținute din analiza statică sunt utilizate pentru detectarea programelor malware și clasificarea familiilor în [Dam and Touili, 2017, Hong et al., 2018, Phan et al., 2018, Hong et al., 2019]. În mod similar, aceste caracteristici sunt utilizate în tehnicile de embedding a nodurilor și grafurilor [Jiang et al., 2018, Hong et al., 2019, Yan et al., 2019] și în metodele GCN [Li et al., 2021]. Este important să menționăm dimensiunea seturilor de date și numărul de clase utilizate în procesul de clasificare. Clasificarea binară simplă (malițios sau nu) este aplicată în [Dam and Touili, 2017, Jiang et al., 2018, Phan et al., 2018, de Oliveira and Sassi, 2021]. [Hong et al., 2018] menționează 7 clase de autori, [Hong et al., 2019] clasifică fișierele în 6 familii, în timp ce [Tang and Qian, 2019] în 9 familii, iar [Yan et al., 2019] menționează 12 familii.

5.2.2 Teoria rețelelor convoluționale grafice

Rețelele convoluționale grafice (GCN) generalizează CNN la structuri de graf utilizând convoluții din teoria spectrală a grafurilor [Kipf and Welling, 2016, Wu et al., 2019, Bruna et al., 2014, Henaff et al., 2015, Chung, 1997].

În această cercetare, folosim modelul GCN introdus în [Kipf and Welling, 2016] bazat pe netezirea Laplaciană, adică calcularea mediei fiecărui punct asupra vecinilor săi din graf. Regula de propagare a acestui GCN spațial este următoarea,

$$\mathbf{H}^{(i+1)} = \sigma\left(\tilde{\mathbf{A}}\mathbf{H}^{(i)}\mathbf{W}^{(i)}\right) \tag{5.1}$$

unde $\mathbf{H}$ reprezintă reprezentarea (încorporată) a datelor, care conține inițial caracteristicile de intrare, și anume $\mathbf{H}^{(0)} = \mathbf{X}$, $\tilde{\mathbf{A}}$ este matricea de adiacență normalizată simetric, $\mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$, care conține bucle proprii, iar $\mathbf{D} = \text{diag}(\mathbf{A} \cdot \mathbf{1})$ este matricea diagonală de grad. Matricea $\mathbf{W}$ reprezintă ponderile rețelei neuronale, iar σ este o funcție de activare neliniară, de obicei ReLU [Goodfellow et al., 2016]. Modelul GCN de mai sus produce un vector de încorporare pentru fiecare nod, prin urmare, acestea trebuie să fie rezumate pentru întregul graf, pentru care o alegere obișnuită este de a utiliza media (avg. pooling).

Învățarea grafurilor conține următoarele trei tipuri de sarcini [Zhou et al., 2020]: (i) sarcini *la nivel de nod*, de exemplu clasificarea nodurilor, (ii) sarcini *la nivel de muchie*, cum ar fi predicția legăturilor, și (iii) sarcini *la nivel de graf*, cum ar fi clasificarea grafurilor.

5.2.3 Clasificarea familiilor utilizând GCN

Graficul de apeluri este combinația dintre graficul fluxului de control și graficul de apeluri de funcții – detalii în [Mester and Bodó, 2021, Mester, 2020]. Caracteristicile la nivel de nod sunt cuvintele de cod LSH utilizate în [Mester and Bodó, 2021].

Am construit un GCN folosind PyTorch, pachetul geometric[1]. Modelul care generează cele mai bune scoruri F_1 este următorul (folosind denumirile oficiale *torch*): trei straturi *GCNConv*, fiecare urmat de *ReLU*, apoi *Dropout*, apoi un strat *GCNConv* urmat de *Dropout* și un strat *global_mean_pool*, în final, un strat complet conectat – pentru probabilitățile de ieșire. Pentru optimizare, utilizăm pierderea *CrossEntropyLoss* și optimizatorul *Adam*, cu o rată de învățare de 0.01.

[1] `https://www.pyg.org`

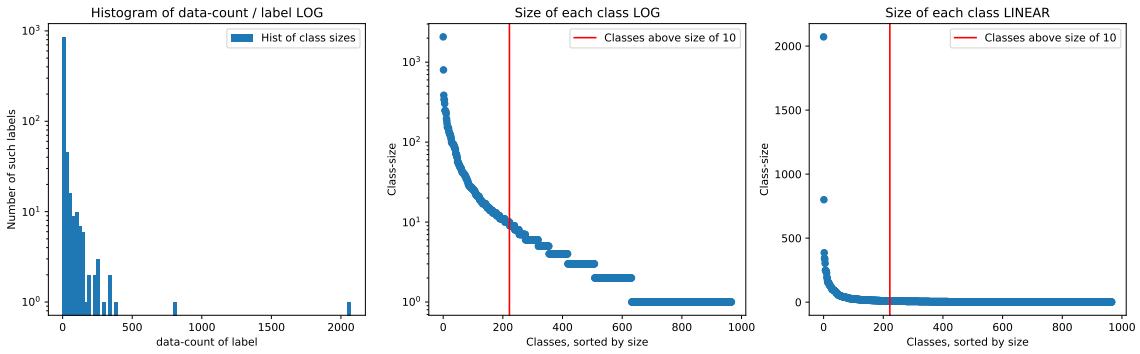


Figura 5.2: Distribuția mărimii familiilor în cadrul setului de date cu $15k$ de fișiere.

5.2.4 Caracteristici la nivel de nod vs. topologie

Experimentele au fost efectuate pe modelul GCN și modelul perceptron multistrat (MLP) cu și fără caracteristici la nivel de nod, de asemenea, și cu alte caracteristici descrise în [Yan et al., 2019].

Prin antrenarea aceluiași model GCN fără caracteristici la nivel de nod (bazat doar pe matricea de adiacență a grafurilor de apel), putem determina dacă cuvintele de cod LSH sunt utile în clasificarea familiilor.

Antrenând un MLP pe caracteristicile la nivel de nod, am putea examina dacă cuvintele de cod LSH pot clasifica familiile de programe malware cu același succes (și anume, scorul F_1) ca modelul GCN complet – cu alte cuvinte, dacă caracteristicile topologice aduc informații valoroase pentru modelul de învățare. În mod similar cu modelul GCN menționat anterior, pentru acest MLP am utilizat funcția de pierdere *CrossEntropyLoss* și optimizatorul *Adam* cu o rată de învățare de 0.01.

În plus, am antrenat aceleași modele GCN/MLP ca înainte, dar folosind un alt set de caracteristici la nivel de nod, cele sugerate în [Yan et al., 2019]. În această lucrare, autorii antrenează un model GCN pe CFG-ul fișierelor, folosind ca caracteristici la nivel de nod un vector de 11 elemente, reprezentând distribuția instrucțiunilor (de exemplu, transfer, apel, aritmetică etc.). În experimentele noastre simulăm această metodă atribuind fiecărei funcții un vector care conține 14 numere, în conformitate cu distribuția mnemonică[2].

[2]x86 Assembly Language Reference: transfer de date, *mov*, transfer de control, *call*, aritmetică, comparare, flag, bit și byte, schimbare și rotire, logică, șir de caractere, I/O; și gradul de intrare și ieșire al funcției respective.

Model	Micro-F_1	Macro-F_1
Model GCN cu coduri LSH	0.381	0.189
Model GCN cu caracteristici de [Yan et al., 2019]	0.614	0.392
Model GCN fără caracteristici la nivel de nod	0.204	0.003
Model MLP cu coduri LSH	0.313	0.050
Model MLP cu caracteristici de [Yan et al., 2019]	0.242	0.020

Tabela 5.1: Scorurile F_1 ale fiecărui model pe setul de date de testare.

5.2.5 Set de date

Setul de date privat Bitdefender conține 15 375 fișiere din 967 de familii. Filtrând familiile cu mai puțin de 10 fișiere, am obținut un set de date de 8620 de fișiere din 223 de familii. Această selecție este ilustrată în Figura 5.2.

5.2.6 Rezultate

Am folosit Python3, IDA Pro 6, GraphViz, PyTorch 1.10.0, Pytorch Geometric 2.0.2, Tensorboard, pe un sistem cu Intel Xeon E5-2697A v4, 64 GB RAM și o placă video GeForce RTX 2080 Ti. Au fost testate diferite combinații de hiperparametri: $1 - 4$ straturi GCN ascunse cu dimensiuni de 64, 128 sau 256, dropout: 0.2, 0.4 sau 0.5.

Cel mai bun model GCN este următorul: 4 straturi GCN de dimensiune 128, dropout de 0.5. Din tabel 5.1 putem concluziona că modelele GCN cu caracteristici la nivel de nod au cele mai bune scoruri F_1, ceea ce înseamnă că utilizarea atât a caracteristicilor topologice ale graficului static de apeluri, cât și a vectorilor de caracteristici ai funcțiilor locale produce cele mai bune rezultate.

5.3 Atribuire cu CNN pe graful static codificat

Pentru o clasificare precisă a familiilor de programe malware, propunem maparea familiilor în funcție de caracteristicile lor comportamentale comune, pe baza modelelor din imaginile lor de instrucțiuni. Traversăm funcțiile graficului de apeluri pentru a compila o listă de instrucțiuni, codificând fiecare instrucțiune într-un pixel pentru a genera imaginea RGB finală. Schița proiectului poate fi văzută în Figura 5.3.

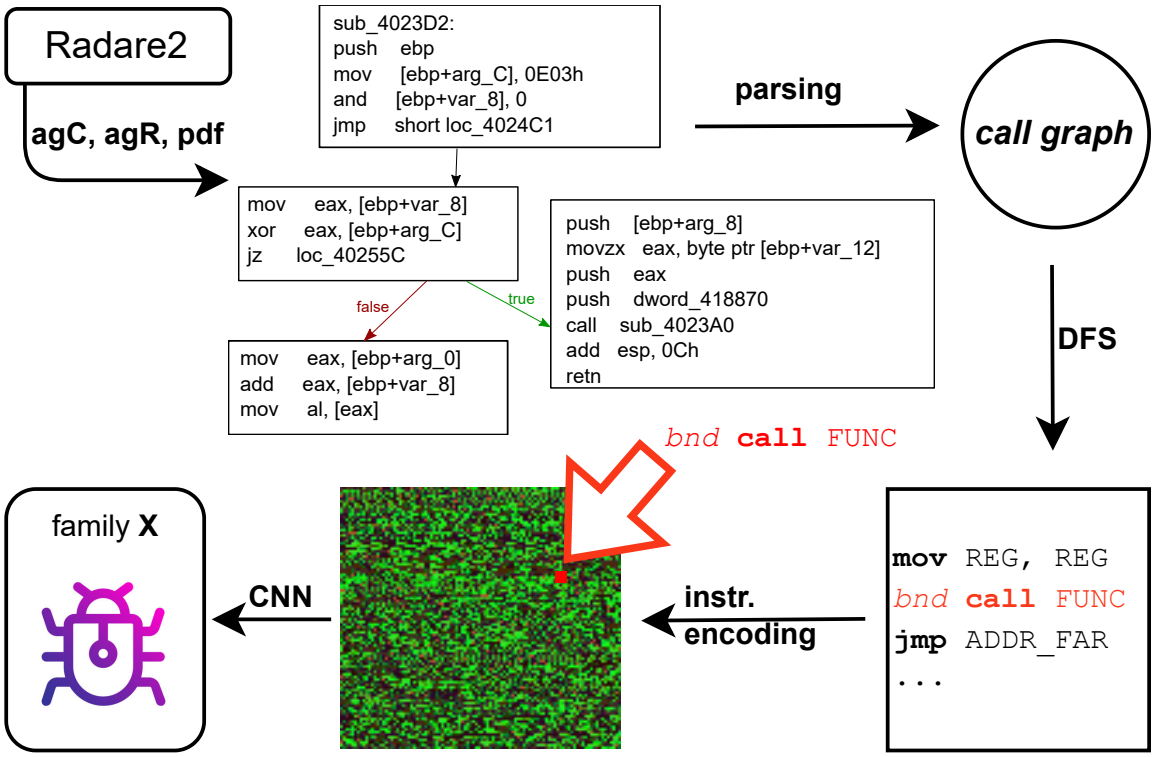


Figura 5.3: Metoda propusă procesează ieșirea Radare2 într-un graf de apeluri, extrage instrucțiunile, le codifică ca o imagine RGB, folosind pentru a antrena CNN-uri.

5.3.1 Analiză a literaturii de specialitate

Primul model care utilizează o reprezentare asemănătoare imaginii a fost descris în [Nataraj et al., 2011], lucrare care prezintă MalImg. Aici se construiește o imagine 2D din binar, în tonuri de gri, fiecare pixel reprezentând un octet. Lucrările [Cui et al., 2018, Kalash et al., 2018] efectuează, de asemenea, experimente folosind reprezentări ale imaginilor în nuanțe de gri ale programelor malware. Diverse alte lucrări de cercetare se bazează pe ideea de a reprezenta codul malware printr-o imagine [Fu et al., 2018, Xiao et al., 2021, Yuan et al., 2020, Deng et al., 2023, Ni et al., 2018]. Imaginile sunt de obicei introduse în rețele neuronale convoluționale (CNN), cele mai populare arhitecturi fiind AlexNet [Krizhevsky et al., 2012], VGG [Simonyan and Zisserman, 2015] și ResNet [He et al., 2016].

5.3.2 Crearea grafului static de apeluri

Generarea graficului static de apeluri necesită un instrument de dezasamblare. Am ales Radare2 (versiunea 5.8.8) din motive detaliate în [Mester, 2023]. În continuare,

enumerăm etapele principale ale generării graficului static de apeluri al unui fișier PE utilizând Radare2 (detalii pe GitHub[3]). Folosind pachetul `r2pipe`[4] aplicăm comenzile `agCd` și `agRd` pentru a obține apelul global și graficul de referință. După parcurgerea acestor grafuri, apelăm pentru fiecare funcție comanda `pdfj` pentru a obține lista de instrucțiuni a acesteia. O instrucțiune conține informații despre flagul bnd [Guide, 2011] și prefixul său (dacă există), mnemonicul său și o listă de parametri. Am luat în considerare 11 prefixe (conform versiunii Radare2 5.8.8) și 7 tipuri de parametri. Structura unei instrucțiuni este următoarea:

[bnd?] [prefix?] mnemonic [param1 [param2, ...]]. (5.2)

5.3.3 Convertirea graficului de apel în imagine

Am intenționat să înțelegem caracteristicile comportamentale statice din graful de apel al unui PE, prin crearea unei imagini bazate pe instrucțiuni. În acest fel, imaginea nu reflectă toți octeții din fiecare secțiune, ci doar instrucțiunile relevante, eliminând astfel *zgomotul*, în comparație cu imaginile în nuanțe de gri bazate pe hex dumps. Aplicăm o traversare DFS a grafului de apeluri pentru a obține lista de funcții, reflectând ordinea lor de execuție. Aplicăm un alt DFS, la nivel de instrucțiuni: în ordinea acestor noduri, de exemplu *A*-*B*-*C*, adunăm instrucțiunile din *A*, dar odată ce întâlnim o instrucțiune de tip *CALL*, sărim la adresa respectivă. După terminarea contextului recursiv, continuăm analizarea instrucțiunilor din blocul *A*. Apoi, trecem la blocul *B* – dacă acesta nu a fost vizitat deja.

Pentru a codifica o instrucțiune într-un pixel (adică o valoare de trei octeți, conform imaginilor RGB), am examinat trei scheme de codificare distincte [Mester et al., 2025]: (i) (FE) Codificare completă: mnemonic, prefix, bnd, doi parametri; (ii) (PE1) Codificare parțială: mnemonic, prefix, bnd; (iii) (PE2) Codificare parțială: doar mnemonic.

5.3.4 Antrenarea modelelor CNN

Folosim modele de ResNet18, ResNet50 [He et al., 2016], ResNet1D [Hong et al., 2020], MobileNetV3 [Howard et al., 2019], GoogleNet [Szegedy et al., 2015], EfficientNet [Tan and Le, 2021] și DenseNet

[3] `https://github.com/attilamester/malflow`
[4] `https://pypi.org/project/r2pipe`

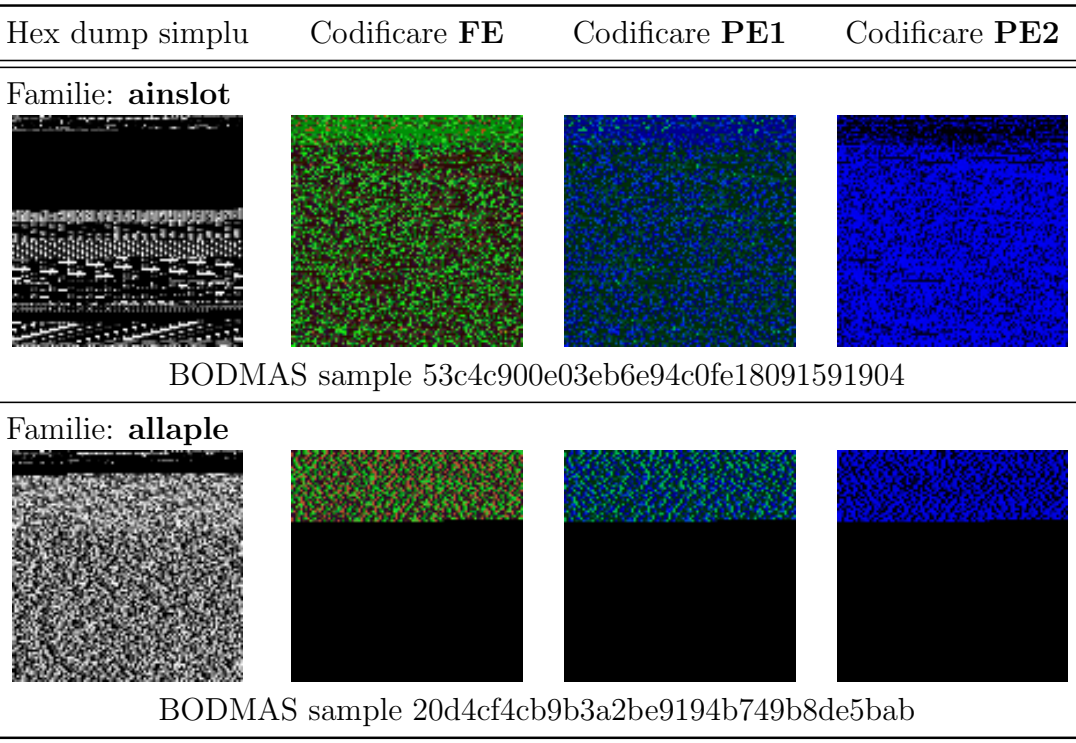


Figura 5.4: Compararea diferitelor codificări ale instrucțiunilor (Section 5.3.3) și a imaginii simple de descărcare hexazecimală pe diferite familii de programe malware.

[Huang et al., 2017]. Experimentele au fost efectuate pe patru tipuri de imagini: imagini în nuanțe de gri bazate pe hex dump (similar cu [Nataraj et al., 2011]) și cele trei tipuri de scheme de codificare a instrucțiunilor prezentate în Secțiunea 5.3.3. În Tabelul 5.3 prezentăm rezultatele conform **FE**, aceasta având cele mai bune rezultate. Am aplicat o căutare aleatorie în spațiul hiperparametrilor: (i) *Arhitectura CNN*; (ii) *Pre-antrenat* (pe ImageNet); (iii) *Numărul minim de fișiere pe clasă* – setat la 100 în tabel 5.3; (iv) *Dimensiunea batch*. Au fost antrenate mai mult de 200 modele, cu timpi de execuție variind de la 30 minute până la 26 ore.

5.3.5 MalImg, BIG, EMBER, BODMAS

Tabelul 5.2 prezintă seturile de date care oferă etichete de familie. MalImg [Nataraj et al., 2011] conține imagini în nuanțe de gri, bazate pe hex dump, de la $9.5k$ de fișiere din 25 de familii [Zhan et al., 2023, Hai et al., 2023, Kim et al., 2023, Moussas and Andreatos, 2021, Gibert et al., 2019]. EMBER (detaliat în Secțiunea 5.3.8) [Anderson and Roth, 2018] este o bază de date semnificativ mai mare decât prima, având vectorul caracteristic de la $800k$ de fișiere

Set de date	Publicat	Binari	Familii	Fișiere(mal.)	EMBER	Disasm.	Image
MalImg	2011	❍	25	9458	❍ ✪	❍ ✪	● ✪
MS BIG	2015	❍	9	10 868	❍	●	❍
EMBER	2018	❍	◗	800 000	●	❍	❍
UCSB-packed	2020	●	❍	232 415	❍	❍	❍
SOREL-20m	2020	●	❍	9 962 820	●	❍	❍
BODMAS	2021	●	581	57 293	●	❍ ✪	❍ ✪
IBD	**2024**	❍	**47**	**18 756**	● ✪	● ✪	● ✪

Tabela 5.2: Seturi de date publice: MalImg [Nataraj et al., 2011], MS BIG [Ronen et al., 2018], EMBER [Anderson and Roth, 2018], UCSB-packed [Aghakhani et al., 2020], SOREL-20m [Harang and Rudd, 2020], BODMAS [Yang et al., 2021] și IBD – contribuția noastră, în cadrul *malflow*. ❍ = "indisponibil", ● = "disponibil", ◗ = "parțial disponibil", ✪ = "publicat de noi".

Model	Batch	Img.	Acc.	F_1 micro	F_1 macro
BODMAS, 57 **de familii**					
ResNet18	20	100×100	0.887	0.887	0.859
BODMAS, 57 **de familii, imagini de tip hex**					
ResNet18	20	100×100	0.881	0.881	0.873
MalImg, 23 **de familii**					
ResNet18	20	100×100	0.722	0.744	0.737
IBD, 47 **de familii**					
ResNet18	20	100×100	0.872	0.872	0.784

Tabela 5.3: Performanța diferitelor modele pe BODMAS, MalImg și IBD.

malițioase [Maillet and Marais, 2023, Jia et al., 2023, Dener and Gulburun, 2023, Sandor et al., 2023, Quertier et al., 2022, Gao et al., 2022, Hussain et al., 2022, Yan et al., 2022, Yang et al., 2021, Feng et al., 2014]. BODMAS [Yang et al., 2021] reprezintă nucleul experimentelor noastre, cu 57 923 de binari, fiecare având o familie și vectorul de caracteristici EMBER.

5.3.6 Set de date

Modelele CNN au fost antrenate pe trei seturi de date diferite: BODMAS, MalImg și un set de date intern Bitdefender (IBD[5]). Figura 5.5 arată distribuția familiilor în cele trei seturi de date, demonstrând dezechilibrul lor ridicat – am încercat, de asemenea, creșterea datelor prin invocarea unui motor metamorfic[6] pentru a genera

[5] `https://kaggle.com/datasets/amester/malflow`

[6] `https://hub.docker.com/repository/docker/attilamester/pymetangine`

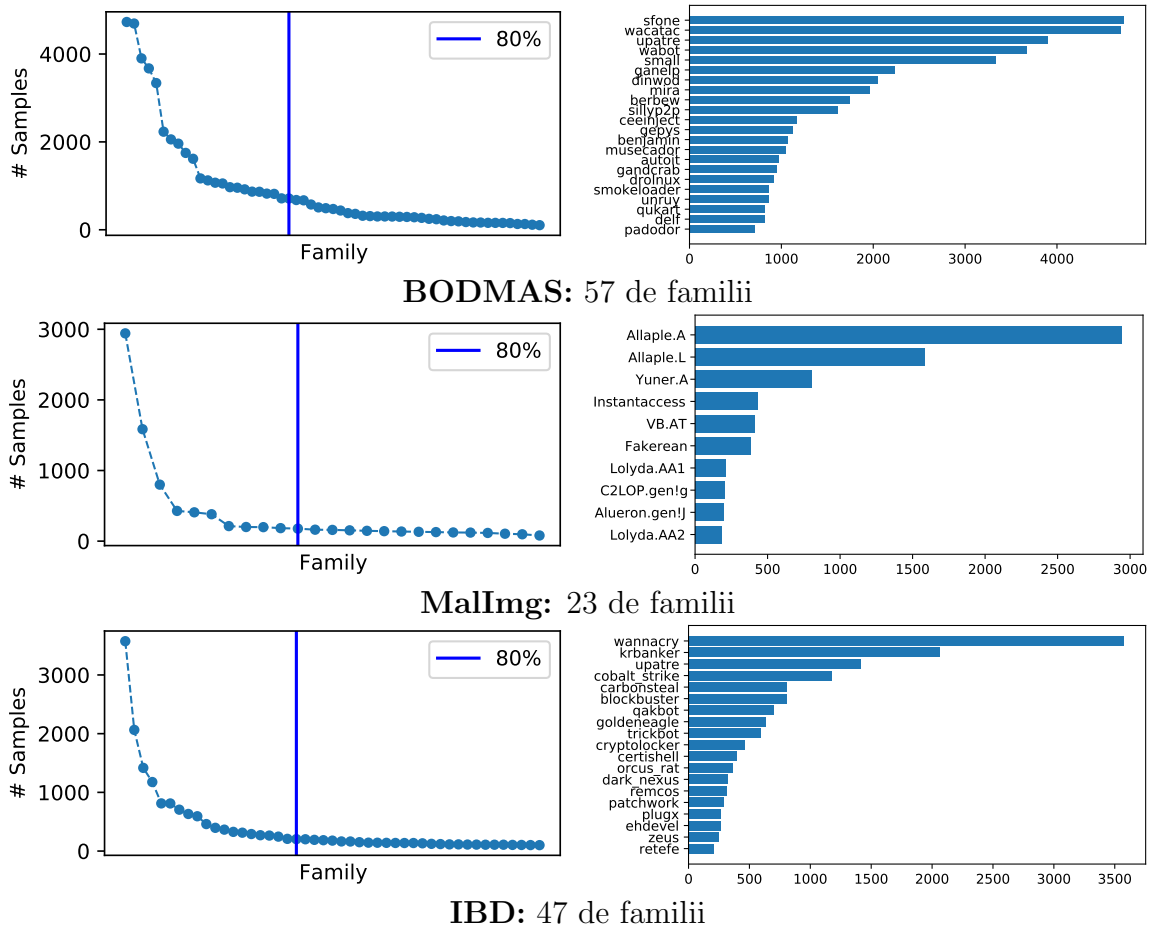


Figura 5.5: Distribuția familiilor în seturile de date și familiile de top care reprezintă 80% din date, utilizate pentru antrenarea CNN-urilor pe imagini de instrucții.

variații pentru familiile mici, fără a obține nici o îmbunătățire semnificativă.

5.3.7 Rezultate

Am folosit Python3.8 cu Radare2 5.8.8 și un server GPU cu două plăci grafice RTX 2080 Ti. Tabelul 5.3 arată rezultatul celor mai bune modele, ResNet18 având cel mai mare scor F_1. Putem conclude că cel mai bun scor F_1 obținut de aceste modele este de 0.887, realizat de un ResNet18 pe 100×100 de imagini, folosind un batch de 20 de imagini, pe setul de date BODMAS și, în mod similar, pe setul de date IBD. Pe setul de date MalImg, câștigătorul a fost tot un model ResNet18, acum cu un scor F_1 de 0.744. Rezultate similare au fost obținute în cazul BODMAS cu imagini de tip hex dump – concluzionând că structura fișierelor, secțiunea de date și alte astfel de "zgomote" sunt încă utile pentru clasificarea familiilor. În mod similar,

informația încapsulată în acest "zgomot", împreună cu corelația sa cu etichetele reale ale familiilor (Capitolul 6) este probabil motivul pentru care clasificarea bazată pe imagini de tip hex dump pe MalImg produce o precizie de 0.98 [Nataraj et al., 2011] în comparație cu scorul mult mai mic de numai 0.72 cu imaginile bazate pe instrucțiuni.

5.3.8 Comparație cu modelul de referință EMBER

EMBER [Anderson and Roth, 2018] conține agregarea caracteristicilor statice dintr-un fișier PE, cum ar fi informațiile de secțiune, import și export[7]. Deoarece experimentele noastre au vizat mai întâi BODMAS [Yang et al., 2021], care conține, de asemenea, caracteristicile EMBER, o bună măsură de comparație a modelului nostru de clasificare a imaginilor pe baza instrucțiunilor grafului de apel a fost transformarea imaginilor în vectori, prin măsurarea distribuției mnemonice a instrucțiunilor (2019 de mnemonice). Pe baza diferitelor modele liniare (arbore de decizie, pădure aleatorie și clasificator SVM), antrenate atât pe seturi de date complete, cât și pe cele filtrate (care conțin numai fișiere neîmpachetate), am ajuns la concluzia că fișierele împachetate denaturează performanța de clasificare în cazul modelului histogramă mnemonică, dar nu și în cazul EMBER, care conține și alte trăsături binare. Am formulat ipoteza că ambalatorii pot fi corelate cu familiile – confirmată în Capitolul 6.

5.4 Concluzii

Acest capitol a prezentat rolul învățării automate supravegheate în contextul clasificării familiilor de programe malware. Am prezentat o abordare bazată pe GCN pentru clasificarea programelor malware, utilizând caracteristici statice bazate pe graficul de apel al unui PE [Mester, 2020, Mester and Bodó, 2021]. Cealaltă metodă prezentată se referă la transformarea graficului static de apeluri al unui PE într-o imagine și utilizarea unui CNN pentru clasificare [Mester et al., 2025] – prezentată la NSS 2024[8]. O altă contribuție este setul de date IBD[9], publicat pe Kaggle[10] – conținând grafuri de apeluri dezasamblate Radare2 din IBD, MalImg și BODMAS, precum și alte informații, cum ar fi ambalatorul, familia, distribuția instrucțiunilor și imaginile RGB codate.

[7] https://github.com/elastic/ember
[8] https://nsclab.org/nss-socialsec2024/papers.html
[9] https://github.com/attilamester/malflow
[10] https://www.kaggle.com/datasets/amester/malflow

Capitolul 6

Împachetare și familii malware

Acest capitol explorează impactul utilizării instrumentelor de împachetare asupra fișierelor executabile, în contextul analizei statice a programelor malware. Am descoperit o corelație între astfel de instrumentele și familiile malware în seturile de date analizate. Împachetarea fișierelor executabile a avut ca scop reducerea utilizării spațiului pe disc. Acesta se realizează prin comprimarea conținutului fișierului și decomprimarea acestuia în timpul rulării – limitând astfel metodele de analiză statică. După cum se menționează în [Aghakhani et al., 2020, Mantovani et al., 2020], un procent semnificativ de fișiere sunt împachetate – confirmat de experimentele noastre.

6.1 Analizarea fișierelor împachetate

Fișierele împachetate pot avea un efect negativ asupra unui clasificator bazat pe caracteristici statice, deoarece codul său binar nu reflectă codul original malițios [Aghakhani et al., 2020, Mantovani et al., 2020] – așa cum am suspectat în Secțiunea 5.3.8. Prin urmare, am examinat dacă există o corelație între astfel de fișiere și familia lor. Corelația a fost măsurată cu ajutorul scorului de asociere Cramér's V [Sheskin, 2000] între trei variabile nominale: familia fișierelor, starea de împachetare și instrumentul, după cum se arată în Tabelul 6.2.

6.2 Concluzii

Acest capitol a investigat impactul împachetării asupra performanței de clasificare a modelului propus în Section 5.3 și a vectorului EMBER. Am arătat că există o

	Total	Împachetat		UPX	Petite	ASPack	DxPack	MPRESS	PEComp
BODMAS	57 293	18 688	(33%)	9 676	3 795	1 771	1 654	1 174	580
MalImg	9 458	1 703	(18%)	1 686	–	–	–	–	4
IBD	18 756	2 195	(12%)	424	159	111	19	214	909

Tabela 6.1: Statistici de împachetare obținute de DIE.

	BODMAS	MalImg	IBD
Familia – Împachetat	0.755	0.981	0.736
Familia – Instrumentul de împachetare	0.547	0.590	0.525

Tabela 6.2: Scorul de asociere Cramér's V între familia malware și ambalatorul.

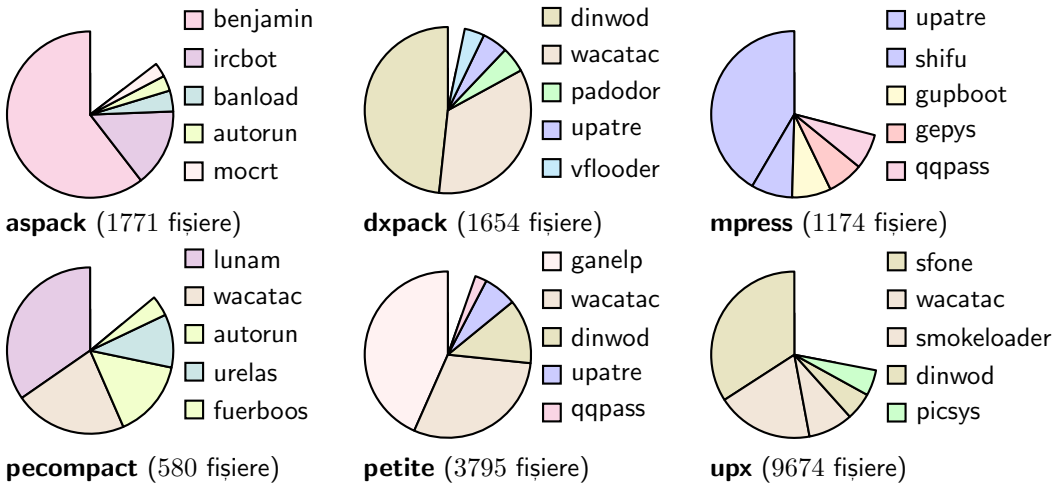


Figura 6.1: Distribuția familiilor pentru instrumentele de împachetare în BODMAS.

corelație între familii și ambalatori în MalImg, BODMAS și IBD, care se întind pe aproape 15 ani și demonstrează că distribuția familie–ambalator corespunde în numeroase cazuri – de exemplu, familia *agent* tinde să fie găsită neîmpachetată atât în BODMAS, cât și în IBD, în timp ce familiile *upatre* nu sunt, în general, împachetate, dar dacă sunt, atunci mai ales cu *mpress* sau *petite*. Figura 6.1 prezintă câteva cazuri interesante – *aspack* este utilizat în principal pentru a împacheta familiile *benjamin* și *ircbot*, în timp ce *petite* este utilizat pentru a împacheta familiile *ganelp* și *wacatac* – în ambele cazuri, în aproximativ 75% din cazuri.

Aceste revelații pot fi utilizate ca punct de plecare pentru analiza familiilor malware – deoarece ambalatorii pot fi determinați într-o manieră relativ programatică, acest lucru poate fi de folos în restrângerea grupului de familii.

Capitolul 7

Concluzii și direcții viitoare

Această teză a contribuit la domeniul analizei și atribuirii programelor malware (și anume clasificarea) prin explorarea diferitelor abordări din literatura de specialitate, oferind o examinare cuprinzătoare a tehnicilor și caracteristicilor utilizate pentru clasificarea programelor malware în familii. Fiecare capitol abordează o metodă diferită aplicabilă în clasificarea familiilor de programe malware, fiecare dintre acestea bazându-se pe ideea de a extrage informații din graficul static de apeluri al unui executabil, obținut cu ajutorul instrumentelor de dezasamblare IDA sau Radare2.

Capitolul 2 introduce concepte-cheie specifice domeniului, inclusiv tipuri de programe malware, metode de analiză și instrumente și caracteristici importante, esențiale pentru investigarea fișierelor malițioase. Capitolul 3 se concentrează pe instrumentele de dezasamblare proeminente IDA și Radare2, dezvăluind funcționalitățile lor scriptate pentru generarea grafurilor de apeluri statice. Comparația detaliată a grafurilor de apeluri generate de aceste instrumente a oferit informații valoroase cu privire la punctele lor forte relative și la aplicabilitatea lor pentru sarcinile de analiză a programelor malware. De asemenea, am motivat utilizarea Radare2 în cercetarea noastră datorită legăturilor sale rapide și fiabile cu Python, care facilitează automatizarea sarcinilor complexe de analiză.

Următoarele două capitole, Capitolul 4 și Capitolul 5 prezintă mai multe metodologii pentru clasificarea familiilor de programe malware – primul utilizează tehnici de învățare nesupravegheate, în timp ce al doilea utilizează învățarea supravegheată. Prezentăm o metodă de grupare a fișierelor malware pe baza semnăturilor sensibile la localitate extrase din graficul lor de apeluri. Aceste semnături vor generaliza comportamentul fișierului, fiind bazate pe instrucțiunile grafurilor, precum și pe structură.

Gruparea se realizează cu ajutorul algoritmilor de detectare a comunităților, care sunt capabili să grupeze fișiere similare. A doua abordare utilizează învățarea supravegheată pentru a clasifica familiile malware. Utilizăm rețele convoluționale grafice pentru a învăța topologia grafului de apeluri și apoi clasificăm fișierele utilizând o rețea neuronală complet conectată. De asemenea, transformăm grafurile de apeluri în imagini și aplicăm mai multe modele de rețele neuronale convoluționale pentru a clasifica fișierele. Ambele abordări au arătat o acuratețe ridicată în clasificarea malware. Direcțiile viitoare pentru aceste abordări includ extinderea metodei hashing sensibile la localitate pentru a include mai multe caracteristici din graficul de apeluri. O altă direcție este de a analiza robustețea modelului de rețea convoluțională grafică construit pe baza grafului de apeluri la atacurile adversarilor și de a-l compara cu alte modele de referință, cum ar fi vectorul de caracteristici EMBER. Cu toate acestea, pentru această analiză, generarea fișierelor adversare va fi cea mai dificilă parte.

Capitolul 6 a abordat impactul packerilor asupra analizei programelor malware, dezvăluind o corelație puternică între packerul utilizat pentru a ofusca un fișier malițios și familia de programe malware asociată acestuia. Analizând trei seturi extinse de date care se întind pe aproape două decenii, am oferit o perspectivă longitudinală asupra relației durabile dintre packeri și familiile de programe malware, oferind perspective valoroase în acest domeniu. Lucrările viitoare sunt necesare pentru a determina metode fiabile de detectare a packerilor, precum și pentru a explora potențialul de utilizare a informațiilor despre packeri pentru a îmbunătăți modelele de clasificare a programelor malware.

Această teză face progrese în domeniul analizei programelor malware prin prezentarea unor abordări inovatoare pentru clasificarea programelor malware, în special la nivel de extragere a caracteristicilor. Metodele prezentate în această teză au demonstrat o acuratețe ridicată în clasificarea fișierelor malware și pot fi utilizate pentru a fi integrate în cadrele de analiză malware. Seturile de date existente, BODMAS și MalImg, sunt analizate prin metodele prezentate în această teză, iar datele prelucrate sunt publicate pe Kaggle[1] pentru a spori transparența activității noastre și a facilita continuarea cercetărilor în domeniu. Codul nostru de procesare a graficului static de apeluri al unui executabil – o caracteristică de bază utilizată în această teză – este, de asemenea, disponibil public pe Github[2].

[1] `https://www.kaggle.com/datasets/amester/malflow`
[2] `https://github.com/attilamester/malflow`

Bibliografie

[Aghakhani et al., 2020] Aghakhani, H., Gritti, F., Mecca, F., Lindorfer, M., Ortolani, S., Balzarotti, D., Vigna, G., and Kruegel, C. (2020). When malware is packin' heat; limits of machine learning classifiers based on static analysis features. In *NDSS*.

[Anderson and Roth, 2018] Anderson, H. S. and Roth, P. (2018). EMBER: an open dataset for training static pe malware machine learning models. arXiv preprint arXiv:1804.04637.

[Andriesse et al., 2016] Andriesse, D., Chen, X., Van Der Veen, V., Slowinska, A., and Bos, H. (2016). An in-depth analysis of disassembly on full-scale x86/x64 binaries. In *USENIX Security Symposium*, pages 583–600.

[Blondel et al., 2008] Blondel, V., Guillaume, J.-L., Lambiotte, R., and Lefebvre, E. (2008). Fast unfolding of communities in large networks. *Journal of Statistical Mechanics Theory and Experiment*, 2008:P10008.

[Bruna et al., 2014] Bruna, J., Zaremba, W., Szlam, A., and LeCun, Y. (2014). Spectral networks and deep locally connected networks on graphs. In *International Conference on Learning Representations (ICLR)*.

[Chung, 1997] Chung, F. R. (1997). *Spectral Graph Theory*. Number 92 in Regional Conference Series in Mathematics. American Mathematical Society.

[Cohen, 2019] Cohen, I. (2019). Deobfuscating apt32 flow graphs with cutter and radare2. Technical report, Check Point Software Technologies.

[Cui et al., 2018] Cui, Z., Xue, F., Cai, X., Cao, Y., Wang, G.-g., and Chen, J. (2018). Detection of malicious code variants based on deep learning. *IEEE Transactions on Industrial Informatics*, 14(7):3187–3196.

[Cunningham et al., 2019] Cunningham, E., Boydell, O., Doherty, C., Roques, B., and Le, Q. (2019). Using text classification methods to detect malware. In *AICS*.

[Dahl et al., 2013] Dahl, G. E., Stokes, J. W., Deng, L., and Yu, D. (2013). Large-scale malware classification using random projections and neural networks. In *ICASSP*, pages 3422–3426. IEEE.

[Dam and Touili, 2017] Dam, K.-H.-T. and Touili, T. (2017). Malware detection based on graph classification. In *Proceedings of the 3rd International Conference on Information Systems Security and Privacy, SCITEPRESS-Science and Technology Publications*.

[de Oliveira and Sassi, 2021] de Oliveira, A. S. and Sassi, R. J. (2021). Behavioral malware detection using deep graph convolutional neural networks. *International Journal of Computer Applications*, 174.

[Dener and Gulburun, 2023] Dener, M. and Gulburun, S. (2023). Clustering-aided supervised malware detection with specialized classifiers and early consensus. *Computers, Materials & Continua*, 75:1235–1251.

[Deng et al., 2023] Deng, H., Guo, C., Shen, G., Cui, Y., and Ping, Y. (2023). MC-TVD: A malware classification method based on three-channel visualization and deep learning. *Computers & Security*, 126:103084.

[Feng et al., 2014] Feng, Y., Anand, S., Dillig, I., and Aiken, A. (2014). Apposcopy: semantics-based detection of android malware through static analysis. In *SIGSOFT*, pages 576–587. ACM.

[Fortunato, 2010] Fortunato, S. (2010). Community detection in graphs. *Physics Reports*, 486(3-5):75–174.

[Fu et al., 2018] Fu, J., Xue, J., Wang, Y., Liu, Z., and Shan, C. (2018). Malware visualization for fine-grained classification. *IEEE Access*, 6:14510–14523.

[Gao et al., 2022] Gao, Y., Hasegawa, H., Yamaguchi, Y., and Shimada, H. (2022). Malware detection by control-flow graph level representation learning with graph isomorphism network. *IEEE Access*, 10:111830–111841.

[Gibert et al., 2020] Gibert, D., Mateu, C., and Planes, J. (2020). The rise of machine learning for detection and classification of malware: Research developments, trends and challenges. *Journal of Network and Computer Applications*, 153:102526.

[Gibert et al., 2019] Gibert, D., Mateu, C., Planes, J., and Vicens, R. (2019). Using convolutional neural networks for classification of malware represented as images. *Journal of Computer Virology and Hacking Techniques*, 15:15–28.

[Goodfellow et al., 2016] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep learning*. MIT Press.

[Guide, 2011] Guide, P. (2011). Intel® 64 and ia-32 architectures software developer's manual. *Volume 3B: System programming Guide, Part*, 2(11):0–40.

[Hai et al., 2023] Hai, T. H., Thieu, V. V., Duong, T. T., Nguyen, H. H., and Huh, E. N. (2023). A proposed new endpoint detection and response with image-based malware detection system. *IEEE Access*, 11:122859–122875.

[Harang and Rudd, 2020] Harang, R. and Rudd, E. M. (2020). Sorel-20m: A large scale benchmark dataset for malicious pe detection. arXiv preprint arXiv:2012.07634.

[Hassen and Chan, 2017] Hassen, M. and Chan, P. K. (2017). Scalable function call graph-based malware classification. In *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*, pages 239–248, Scottsdale, AZ, USA. ACM.

[He et al., 2016] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *CVPR*, pages 770–778.

[Henaff et al., 2015] Henaff, M., Bruna, J., and LeCun, Y. (2015). Deep convolutional networks on graph-structured data. arXiv preprint arXiv:1506.05163.

[Hong et al., 2018] Hong, J., Park, S., Kim, S.-W., Kim, D., and Kim, W. (2018). Classifying malwares for identification of author groups. *Concurrency and Computation: Practice and Experience*, 30(3):e4197.

[Hong et al., 2019] Hong, J., Park, S.-J., Kim, T., Noh, Y.-K., Kim, S.-W., Kim, D., and Kim, W. (2019). Malware classification for identifying author groups: a graph-based approach. In *RACS*, pages 169–174. ACM.

[Hong et al., 2020] Hong, S., Xu, Y., Khare, A., Priambada, S., Maher, K., Aljiffry, A., Sun, J., and Tumanov, A. (2020). Holmes: Health online model ensemble serving for deep learning models in intensive care units. In *SIGKDD*, pages 1614–1624.

[Howard et al., 2019] Howard, A., Sandler, M., Chu, G., Chen, L.-C., Chen, B., Tan, M., Wang, W., Zhu, Y., Pang, R., Vasudevan, V., et al. (2019). Searching for mobilenetv3. In *ICCV*, pages 1314–1324.

[Huang et al., 2017] Huang, G., Liu, Z., Van Der Maaten, L., and Weinberger, K. Q. (2017). Densely connected convolutional networks. In *CVPR*, pages 4700–4708.

[Hussain et al., 2022] Hussain, M. J., Shaoor, A., Baig, S., Hussain, A., and Muqurrab, S. A. (2022). A hierarchical based ensemble classifier for behavioral malware detection using machine learning. In *IBCAST*, pages 702–706.

[Jia et al., 2023] Jia, L., Yang, Y., Tang, B., and Jiang, Z. (2023). ERMDS: A obfuscation dataset for evaluating robustness of learning-based malware detection system. *BenchCouncil Transactions on Benchmarks, Standards and Evaluations*, 3:100106.

[Jiang et al., 2018] Jiang, H., Turki, T., and Wang, J. T. (2018). DLGraph: Malware detection using deep learning and graph embedding. In *ICMLA*, pages 1029–1033. IEEE.

[Kalash et al., 2018] Kalash, M., Rochan, M., Mohammed, N., Bruce, N. D., Wang, Y., and Iqbal, F. (2018). Malware classification with deep convolutional neural networks. In *NTMS*, pages 1–5. IEEE.

[Kilgallon et al., 2017] Kilgallon, S., De La Rosa, L., and Cavazos, J. (2017). Improving the effectiveness and efficiency of dynamic malware analysis with machine learning. In *2017 Resilience Week (RWS)*, pages 30–36.

[Kim et al., 2023] Kim, J., Paik, J. Y., and Cho, E. S. (2023). Attention-Based Cross-Modal CNN Using Non-Disassembled Files for Malware Classification. *IEEE Access*, 11:22889–22903.

[Kipf and Welling, 2016] Kipf, T. N. and Welling, M. (2016). Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907.

[Koo et al., 2021] Koo, H., Park, S., and Kim, T. (2021). A look back on a function identification problem. In *Annual Computer Security Applications Conference*, pages 158–168.

[Krizhevsky et al., 2012] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. In *NeurIPS*.

[Li et al., 2021] Li, S., Zhou, Q., Zhou, R., and Lv, Q. (2021). Intelligent malware detection based on graph convolutional network. *The Journal of Supercomputing*, pages 1–17.

[Maillet and Marais, 2023] Maillet, W. and Marais, B. (2023). Neural networks optimizations against concept and data drift in malware detection. arXiv preprint arXiv:2308.10821.

[Mantovani et al., 2020] Mantovani, A., Aonzo, S., Ugarte-Pedrero, X., Merlo, A., and Balzarotti, D. (2020). Prevalence and impact of low-entropy packing schemes in the malware ecosystem. In *NDSS*.

[Martinelli et al., 2017] Martinelli, F., Marulli, F., and Mercaldo, F. (2017). Evaluating convolutional neural network for effective mobile malware detection. *Procedia Computer Science*, 112:2372–2381.

[Massarelli et al., 2019] Massarelli, L., Di Luna, G. A., Petroni, F., Baldoni, R., and Querzoni, L. (2019). Safe: Self-attentive function embeddings for binary similarity. In *Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 309–329, Cham. Springer International Publishing.

[McLaughlin et al., 2017] McLaughlin, N., Martinez del Rincon, J., Kang, B., et al. (2017). Deep Android malware detection. In *CODASPY*, pages 301–308. ACM.

[Mester, 2020] Mester, A. (2020). Scalable, real-time malware clustering based on signatures of static call graph features. Master's thesis, Babeş–Bolyai University, Faculty of Mathematics and Computer Science, Cluj-Napoca, Romania.

[Mester, 2023] Mester, A. (2023). Malware analysis and static call graph generation with Radare2. *Studia Universitatis Babeș-Bolyai Informatica*, 68(1):5–20.

[Mester et al., 2025] Mester, A., Bodó, Z., Vinod, P., and Conti, M. (2025). Towards a malware family classification model using static call graph instruction visualization. In *Network and System Security*, pages 167–186, Singapore. Springer Nature Singapore.

[Mester and Bodó, 2021] Mester, A. and Bodó, Z. (2021). Validating static call graph-based malware signatures using community detection methods. In *ESANN*, pages 429–434.

[Mester and Bodó, 2022] Mester, A. and Bodó, Z. (2022). Malware classification based on graph convolutional neural networks and static call graph features. In *IEA/AIE*, pages 528–539. Springer.

[Mester et al., 2021] Mester, A., Pop, A., Mursa, B.-E.-M., Greblă, H., Dioşan, L., and Chira, C. (2021). Network analysis based on important node selection and community detection. *Mathematics*, 9(18):2294.

[Moussas and Andreatos, 2021] Moussas, V. and Andreatos, A. (2021). Malware detection based on code visualization and two-level classification. *Information*, 12:1–14.

[Nar et al., 2019] Nar, M., Kakisim, A. G., Yavuz, M. N., and Soğukpinar, İ. (2019). Analysis and comparison of disassemblers for opcode based malware analysis. In *2019 4th International Conference on Computer Science and Engineering (UBMK)*, pages 17–22. IEEE.

[Nataraj et al., 2011] Nataraj, L., Karthikeyan, S., Jacob, G., and Manjunath, B. S. (2011). Malware images: visualization and automatic classification. In *VizSec*, pages 1–7.

[Ni et al., 2018] Ni, S., Qian, Q., and Zhang, R. (2018). Malware identification using visualization images and deep learning. *Computers & Security*, 77:871–885.

[Oprisa et al., 2014] Oprisa, C., Checiches, M., and Nandrean, A. (2014). Locality-sensitive hashing optimizations for fast malware clustering. In *Proceedings of the 10th International Conference on Intelligent Computer Communication and Processing (ICCP)*, pages 97–104, Cluj-Napoca, Romania. IEEE.

[Park et al., 2010] Park, Y., Reeves, D., Mulukutla, V., and Sundaravel, B. (2010). Fast malware classification by automated behavioral graph matching. In *CSIIRW*, pages 1–4.

[Phan et al., 2018] Phan, A. V., Le Nguyen, M., Nguyen, Y. L. H., and Bui, L. T. (2018). DGCNN: A convolutional neural network over large-scale labeled graphs. *Neural Networks*, 108:533–543.

[Priyanga et al., 2022] Priyanga, S., Suresh, R., Romana, S., and Shankar Sriram, V. (2022). The good, the bad, and the missing: A comprehensive study on the rise of machine learning for binary code analysis. In *Computational Intelligence in Data Mining: Proceedings of ICCIDM 2021*, pages 397–406. Springer.

[Quertier et al., 2022] Quertier, T., Marais, B., Morucci, S., and Fournel, B. (2022). MERLIN – Malware Evasion with Reinforcement LearnINg. arXiv preprint arXiv:2203.12980.

[Radare, 2009] Radare (2009). The official radare2 book. https://book.rada.re/.

[Ronen et al., 2018] Ronen, R., Radu, M., Feuerstein, C., Yom-Tov, E., and Ahmadi, M. (2018). Microsoft malware classification challenge. arXiv preprint arXiv:1802.10135.

[Sandor et al., 2023] Sandor, M., Portase, R. M., and Colesa, A. (2023). Ember feature dataset analysis for malware detection. In *ICCP*, pages 203–210.

[Shaila et al., 2021] Shaila, S., Darki, A., Faloutsos, M., Abu-Ghazaleh, N., and Sridharan, M. (2021). Disco: Combining disassemblers for improved performance. In *Proceedings of the 24th International Symposium on Research in Attacks, Intrusions and Defenses*, pages 148–161.

[Sheskin, 2000] Sheskin, D. J. (2000). *Handbook of parametric and nonparametric statistical procedures*. Chapman & Hall/CRC.

[Simonyan and Zisserman, 2015] Simonyan, K. and Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556.

[Steffens, 2020] Steffens, T. (2020). *Attribution of Advanced Persistent Threats*. Springer.

[Szegedy et al., 2015] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015). Going deeper with convolutions. In *CVPR*, pages 1–9.

[Tahir, 2018] Tahir, R. (2018). A study on malware and malware detection techniques. *International Journal of Education and Management Engineering*, 8(2):20.

[Tan and Le, 2021] Tan, M. and Le, Q. (2021). EfficientNetV2: Smaller models and faster training. In *ICML*, pages 10096–10106. PMLR.

[Tang and Qian, 2019] Tang, M. and Qian, Q. (2019). Dynamic api call sequence visualisation for malware classification. *IET Information Security*, 13(4):367–377.

[Topan et al., 2013] Topan, V. I., Dudea, S. V., and Canja, V. D. (2013). Fuzzy whitelisting anti-malware systems and methods. US Patent 8,584,235.

[Ucci et al., 2019] Ucci, D., Aniello, L., and Baldoni, R. (2019). Survey of machine learning techniques for malware analysis. *Computers & Security*, 81:123–147.

[Wenzl et al., 2019] Wenzl, M., Merzdovnik, G., Ullrich, J., and Weippl, E. (2019). From hack to elaborate technique—a survey on binary rewriting. *ACM Computing Surveys (CSUR)*, 52(3):1–37.

[Wu et al., 2019] Wu, F., Souza, A., Zhang, T., Fifty, C., Yu, T., and Weinberger, K. (2019). Simplifying graph convolutional networks. In *International Conference on Machine Learning*, pages 6861–6871. PMLR.

[Xiao et al., 2021] Xiao, M., Guo, C., Shen, G., Cui, Y., and Jiang, C. (2021). Image-based malware classification using section distribution information. *Computers & Security*, 110:102420.

[Yan et al., 2022] Yan, J., Jia, X., Ying, L., Yan, J., and Su, P. (2022). Understanding and mitigating label bias in malware classification: An empirical study. In *QRS*, pages 492–503.

[Yan et al., 2019] Yan, J., Yan, G., and Jin, D. (2019). Classifying malware represented as control flow graphs using deep graph convolutional neural network. In *DSN*, pages 52–63. IEEE.

[Yang et al., 2021] Yang, L., Ciptadi, A., Laziuk, I., Ahmadzadeh, A., and Wang, G. (2021). BODMAS: An Open Dataset for Learning based Temporal Analysis of PE Malware. In *SPW*, pages 78–84.

[Yin et al., 2018] Yin, X., Liu, S., Liu, L., and Xiao, D. (2018). Function recognition in stripped binary of embedded devices. *IEEE Access*, 6:75682–75694.

[Yuan et al., 2020] Yuan, B., Wang, J., Liu, D., Guo, W., Wu, P., and Bao, X. (2020). Byte-level malware classification based on markov images and deep learning. *Computers & Security*, 92:101740.

[Zhan et al., 2023] Zhan, D., Duan, Y., Hu, Y., Yin, L., Pan, Z., and Guo, S. (2023). AMGmal: Adaptive mask-guided adversarial attack against malware detection with minimal perturbation. *Computers & Security*, 127.

[Zhou et al., 2020] Zhou, J., Cui, G., Hu, S., Zhang, Z., Yang, C., Liu, Z., Wang, L., Li, C., and Sun, M. (2020). Graph neural networks: A review of methods and applications. *AI Open*, 1:57–81.