UNIVERSITATEA TEHNICĂ „Gheorghe Asachi” din IAȘI
FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE
DOMENIUL: Calculatoare și Tehnologia Informației
SPECIALIZAREA: Tehnologia Informației

LUCRARE DE DIPLOMĂ

Coordonator științific:
Ș.l.dr.ing. Cătălin Mironeanu

Absolvent:
Ștefan Paraschiv

Iași, 2023

UNIVERSITATEA TEHNICĂ „Gheorghe Asachi” din IAȘI
FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE
DOMENIUL: Calculatoare și Tehnologia Informației
SPECIALIZAREA: Tehnologia Informației

Analiza comportamentală a fișierelor potențial malițioase în cadrul unui mediu izolat utilizând o arhitectură modulară

LUCRARE DE DIPLOMĂ

Coordonator științific:
Ș.l.dr.ing. Cătălin Mironeanu

Absolvent:
Ștefan Paraschiv

Iași, 2023

Cuprins

Introducere **1**

1 Fundamentarea teoretică și documentarea bibliografică **3**
1.1 Domeniul și contextul abordării temei . . . 3
1.2 Fundament teoretic . . . 3
1.2.1 Malware . . . 3
1.2.2 Sandbox . . . 5
1.2.3 Windows API . . . 5
1.2.4 Reguli Yara . . . 7
1.2.5 Corelarea datelor într-o platformă de tip Threat Intelligence . . . 7
1.2.6 Învățare automată . . . 7
1.3 Referințe la subiecte similare . . . 9
1.3.1 Tehnologii de sandboxing . . . 10
1.3.1.1 Cuckoo . . . 10
1.3.1.2 ANY.RUN . . . 11
1.3.1.3 FortiSandbox . . . 12
1.3.2 Soluții similare de malware analysis . . . 13
1.3.2.1 Detecție de malware prin creare de reguli bazate pe mai multe caracteristici . . . 14
1.3.2.2 Detecție de malware bazată pe apeluri API și analiză comportamentală . . . 15
1.3.2.3 Clasificare de malware bazată pe apeluri API și analiză comportamentală . . . 16
1.3.2.4 Studiu de detecție și clasificare de malware bazată pe apeluri API utilizând rețele neuronale recurente și algoritmi tradiționali de învățare automată . . . 17
1.3.2.5 Clasificare de malware bazată pe vizualizarea secvențelor API . . . 17
1.4 Metoda propusă . . . 18

2 Proiectarea aplicației **19**
2.1 Arhitectura aplicației . . . 19
2.2 Modulul de extragere a atașamentelor din e-mail . . . 20
2.3 Modulul Cuckoo . . . 21
2.4 Modulul MISP . . . 22
2.5 Modulul de extragere de caracteristici . . . 23
2.5.1 Extragere de caracteristici pentru reguli Yara . . . 24
2.5.2 Extragere de caracteristici pentru model . . . 26
2.6 Modulul de antrenare a modelelor . . . 27
2.7 Modulul de generare a regulilor Yara . . . 28
2.8 Modulul de potrivire cu reguli Yara . . . 30

2.9 Modulul de clasificare 30

3 Implementarea aplicației **31**
3.1 Modulul de extragere a atașamentelor din e-mail 31
3.2 Modulul Cuckoo 32
3.3 Modulul MISP 32
3.4 Modulul de extragere de caracterisitici 32
3.4.1 Extragere de caracterisitici pentru reguli Yara 32
3.4.2 Extragere de caracterisitici pentru model 35
3.5 Modulul de antrenare și evaluare al modelelor 35
3.6 Modulul de generare a regulilor Yara 36
3.7 Modulul de potrivire cu reguli Yara 37
3.8 Modulul de clasificare 38

4 Testarea aplicației și rezultate experimentale **39**
4.1 Punerea în funcțiune a aplicației 39
4.2 Rezultate experimentale 40
4.2.1 Rezultate modul Yara 40
4.2.2 Rezultate Cuckoo 44
4.3 Cazuri de utilizare a aplicației 45

Concluzii **47**

Bibliografie **49**

Analiza comportamentală a fișierelor potențial malițioase în cadrul unui mediu izolat utilizând o arhitectură modulară

Ștefan Paraschiv

Rezumat

Evoluția continuă a amenințărilor malware și utilizarea tehnicilor avansate de atac, cum ar fi ofuscarea codului (în sensul termenului în limba engleză *obfuscation*), exercită o presiune semnificativă asupra securității infrastructurilor IT, a organizațiilor și a indivizilor. Pentru a face față acestei amenințări în continuă evoluție este importantă dezvoltarea unor metode de detectare timpurie a malware-ului. Aplicația propusă vine în întâmpinarea acestei nevoi, oferind o soluție de sprijin pentru asigurarea unui mediu de lucru protejat.

Lucrarea propune o arhitectură modulară pentru identificarea fișierelor malițioase prin analiză hibridă. Această abordare se bazează pe analiza apelurilor de sistem, cu date care pot fi vehiculate prin intermediul unui serviciu de e-mail. În continuare, aceste date sunt monitorizate și înregistrate într-un raport generat prin rularea fișierelor într-un mediu controlat de tip *sandbox*. Raportul generat în urma acestei analize este ulterior procesat de două module proprii, iar în același timp este trimis către platforma MISP - un sistem dedicat stocării și corelării informațiilor de securitate.

Modulele dezvoltate se bazează pe reguli Yara și un model de învățare automată, care permit determinarea naturii și caracteristicilor fișierelor analizate. Aceste module utilizează ca atribute principale pentru clasificare și generare de reguli Yara, apeluri API care sunt procesate în două modalități distincte.

Pentru generarea regulilor Yara, aplicația utilizează algoritmii TF-IDF (*Term Frequency-Inverse Document Frequency*) și LCS (*Longest Common Subsequence*) în contextul analizei atributelor precum apeluri API, adrese IP și DNS. Algoritmul TF-IDF poate fi aplicat asupra acestor atribute pentru a genera reguli care ulterior sunt utilizate în procesul de clasificare. Algoritmul LCS este folosit pentru identificarea secvențelor suspecte de apeluri API din raportul generat. Prin combinarea acestor algoritmi, aplicația poate genera reguli Yara, care contribuie la detectarea și clasificarea malware-ului.

Modulul de învățare automată, dezvoltat în cadrul aplicației utilizează secvențe non-consecutive de apeluri API de lungime o sută care sunt extrase din raportul generat de Cuckoo. După extragerea acestor secvențe, a fost efectuată o analiză comparativă a patru modele de clasificare (Random Forest, Logistic Regression, Support Vector Machine și K-Nearest Neighbor) pentru a evalua și clasifica fișierele analizate. Din acestea, în cazul particular al datelor prelucrate, prin utilizarea modelului Random Forest s-au obținut cele mai bune rezultate de clasificare.

Pentru aplicarea soluției într-un scenariu realistic, este efectuată integrarea cu un serviciu de e-mail. Astfel, datele furnizate către *sandbox* reprezentă atașamentele unui e-mail, care sunt preluate și trimise modulelor proiectate și implementate. După finalizarea analizei, utilizatorul primește un răspuns individual de la fiecare modul de analiză, furnizându-i un verdict în legătură cu fișierul analizat.

Introducere

Una din funcțiile aplicațiilor Industriei 4.0 este securitatea cibernetică. Aceasta asigură implementarea măsurilor de securitate pentru a minimiza amenințările cibernetice externe și interne. Această funcție face parte din domeniul industriei IT, iar acest domeniu este caracterizat de o evoluție rapidă și continuă. Odată cu această evoluție apar și numeroase amenințări, ceea ce face ca securitatea să fie un aspect crucial pentru orice organizație care operează în acest domeniu. Luarea unor măsuri eficiente și potrivite de protejare împotriva atacurilor implică înțelegerea acestui domeniu. Chiar dacă securitatea absolută nu poate fi garantată, este esențial să fie luate măsuri de precauție/prevenție pentru a asigura protecția împotriva potențialelor amenințări.

Lucrarea propune crearea unui mediu pentru analizarea fișierelor prin intermediul unui *sandbox* specializat pe securitate cu rezultate obținute prin algoritmi, tehnici de *machine learning* sau prin alte instrumente. Toate acestea vor fi centralizate într-o bază de date specializată pentru stocarea informațiilor legate de securitatea cibernetică. Prin aceasta se obține posibilitatea explorării domeniului, identificarea potențialelor amenințări și reducerea riscurilor de securitate în industria IT.

În general, majoritatea organizațiilor se bazează exclusiv pe un software antivirus pentru a asigura protecția împotriva amenințărilor cibernetice. Această abordare poate reprezenta un dezavantaj deoarece un astfel de software este în general eficient în detectarea atacurilor cunoscute. Pe de altă parte, un program anti-malware analizează fișierele și codul pentru a detecta modele cunoscute și semnături de malware, viruși și alte amenințări. De obicei, acesta funcționează în timp real, scanând fișierele pe măsură ce acestea sunt accesate, create sau modificate pe un dispozitiv. Unele produse din această categorie au capabilitatea de a detecta și bloca adresele URL rău intenționate, încercările de *Phising* și alte tipuri de atacuri. O alternativă pentru programele antivirus este *sandboxing*-ul, care reprezintă o tehnică utilizată pentru a analiza și detecta malware nou și, implicit, necunoscut.

Un *sandbox* se definește ca un mediu izolat și controlat care este utilizat în principal pentru executarea unui cod nesigur, permițând analizarea comportamentului și detectarea amenințărilor. Acest lucru este posibil deoarece procesele și programele care rulează într-un *sandbox* sunt restricționate să nu interacționeze cu sistemul gazdă. În cadrul acestei lucrări este propusă utilizarea unui *sandbox* pentru analizarea malware-urilor prin combinarea mai multor soluții existente de detecție a fișierelor rău intenționate.

Unul dintre avantajele utilizării unui *sandbox* constă în posibilitatea de a adăuga module specializate care să ajute la analiza datelor – un exemplu ar fi integrarea unui antivirus cu un *sandbox* pentru detectarea avansată a amenințărilor. Astfel, lucrarea propune utilizarea unui *sandbox* cu următoarele module: MISP pentru corelarea automată a datelor, Gmail pentru a demonstra aplicabilitatea în lumea reală și două module proprii pentru a analiza comportamentul unui fișier și a determina dacă acesta prezintă un comportament suspect sau nu. Dintre modulele proprii, primul are rolul de a genera reguli bazate pe apeluri de sistem și trafic de internet care vor determina dacă fișierele sunt malițioase, iar al doilea modul analizează doar apeluri de sistem prin algoritmi de *machine learning* pentru a determina natura fișierului. Astfel verdictul final se face prin combinarea rezultatelor celor două module.

Lucrarea este structurată în patru capitole. În primul capitol se prezintă informații teoretice relevante pentru realizarea proiectului. De asemenea, sunt prezentate și soluții existente din domeniul temei abordate, cu avantajele și dezavantajele fiecăreia.

Capitolul al doilea este dedicat prezentării arhitecturii proiectului, inclusiv descrierea modului de interacțiune între modulele utilizate în *sandbox*. Totodată, sunt analizate avantajele și dezavantajele acestei arhitecturi prin intermediul diagramelor și a metodelor de prezentare vizuală.

În al treilea capitol al lucrării sunt prezentate detaliile privind implementarea proiectului, incluzând pașii urmați în procesul de dezvoltare și descrierea modului de funcționare a sistemului creat. De asemenea, sunt abordate problemele întâmpinate pe parcursul dezvoltării, cum ar fi erorile de implementare sau dificultățile de configurare a anumitor module.

Capitolul patru al lucrării este dedicat testării sistemului propus. Acest capitol începe prin a prezenta scenariile de testare și mediul în care se realizează. Apoi, sunt prezentate rezultatele experimentale obținute în urma testării sistemului, inclusiv datele de test și metricile utilizate pentru a evalua performanța sistemului.

Capitolul 1. Fundamentarea teoretică și documentarea bibliografică

1.1. Domeniul și contextul abordării temei

În prezent, prevenirea infectării unui sistem este o problemă critică iar mulți utilizatori se bazează pe un antivirus pentru a se proteja împotriva potențialelor amenințări. Cu toate acestea, un program antivirus are dezavantajul de a nu detecta precis atacuri noi. Pe de altă parte, tehnologia *sandbox* oferă o soluție prin care se pot detecta și analiza amenințările necunoscute prin izolarea lor într-un mediu controlat și sigur [1].

În contextul amenințărilor de securitate, nu este suficient doar să fie detectate și să fie eliminate amenințările malware, fiind la fel de important să se înțeleagă modul de operare al acestor amenințări, obiectivele lor și contextul în care acționează. Malware, prescurtarea de la software rău intenționat (*malicious software*), se referă la orice software dezvoltat de către infractori cibernetici (numiți impropriu hackeri) pentru a fura date și pentru a afecta sau distruge date stocate în sisteme informatice. Din categoria malware comun fac parte viruși, Ransomware, viermi, Trojan și Spyware [2], după cum este sintetizat în Figura 1.1.

Un avantaj a tehnologiei *sandbox* constă în flexibilitatea sa extinsă în ceea ce privește numărul de componente și funcționalități pe care le poate avea. Această flexibilitate permite adaptarea mediului creat în *sandbox* în funcție de nevoile și cerințele specifice ale analizei sau aplicației în cauză. În plus, se pot adăuga module specializate care să ajute la analizarea datelor și să permită compararea rezultatelor. Multe beneficii derivă din acest aspect deoarece permite utilizatorului să construiască un mediu complex și adaptat precis nevoilor sale pentru a dezvolta un produs de securitate. De asemenea, acest lucru permite analiza separată a diferitelor aspecte, cum ar fi: traficul de internet cu ajutorul instrumentelor dedicate precum Suricata, procese, apeluri de sistem, în timp ce *sandbox*-ul poate oferi mai puține posibilități de analiză individuală în rezultatele sale.

Un alt avantaj semnificativ este reprezentat de existența posibilității de a construi module proprii pentru analiza rezultatelor furnizate de *sandbox*, având în vedere că, de exemplu, utilizarea învățării automate este considerată un element cheie pentru protejarea clienților împotriva amenințărilor. Astfel, se pot integra o varietate de module bazate pe diferiți algoritmi și de învățare automată, care ar putea oferi o analiză mai precisă și mai detaliată. De asemenea, se pot compara rezultatele obținute cu cele furnizate de *sandbox*, pentru a determina dacă modulul construit este mai eficient decât soluțiile existente sau decât alte module proprii implementate.

Lucrarea de diplomă își propune utilizarea unui *sandbox* care va funcționa cu module personalizate și instrumente existente, în vederea determinării intențiilor unui fișier pe baza apelurilor API (eng. *Application Programming Interface*).

1.2. Fundament teoretic

1.2.1. Malware

Un atac cibernetic[1] este caracterizat prin acces neautorizat la sistem/rețea care urmărește să deterioreze, să fure sau să modifice în secret date. Acestea sunt de mai multe tipuri, printre cele mai populare sunt din categoria malware, bazate pe tehnici de phishing, MITM (Man-in-the-middle-attack), DoS/DDoS și SQL injection. Aceste tipuri se diferențiază prin modalitatea în care o victima este infectată. Luând ca exemplu cea mai mai răspândită tehnică – phishing-ul – are ca scop principal replicarea, impersonarea unei entități sau a unei persoane de încredere, cu scopul de a manipula utilizatorul și de a-l infecta sau să divulge informații personale și confidențiale.

Malware este un termen general [3] (provenit din cuvintele din limba engleză „software”

[1] https://nordvpn.com/ro/cybersecurity/

și „malicious”) folosit pentru a descrie un program sau cod creat pentru a dăuna unor dispozitive conectate la rețea (de la servere, stații de lucru, până la dispozitive IoT). Criminalii cibernetici dezvoltă malware pentru a se infiltra într-un sistem informatic, pentru a copia sau distruge date sensibile din aceste sisteme. Malware sunt împărțite în categorii/familii. În Figura 1.1 sunt prezentate cele mai relevante familii și modul în care acestea operează.

Type	What It Does	Real-World Example
Ransomware	Disables victim's access to data until ransom is paid	RYUK
Fileless Malware	Makes changes to files that are native to the OS	Astaroth
Spyware	Collects user activity data without their knowledge	DarkHotel
Adware	Serves unwanted advertisements	Fireball
Trojans	Disguises itself as desirable code	Emotet
Worms	Spreads through a network by replicating itself	Stuxnet
Rootkits	Gives hackers remote control of a victim's device	Zacinlo
Keyloggers	Monitors users' keystrokes	Olympic Vision
Bots	Launches a broad flood of attacks	Echobot
Mobile Malware	Infects mobile devices	Triada
Wiper Malware	Erases user data beyond recoverability.	WhisperGate

Figura 1.1. Familii malware, conform Crowdstrike®[2].

Fiecare familie de malware are anumite trăsături specifice. În cazul tehnicii Ransomware, bazată pe CryptoLocker, este caracterizată prin faptul că înlătură accesul utilizatorului la date. În literatura de specialitate există lucrări care au propus diferite abordări pentru detecție sau clasificarea în familii de malware prin diferite metode. Pentru această lucrare familia de malware care a fost aleasă pentru a fi detectată este Ransomware-ul deoarece este o categorie reprezentativă de prezintă de malware, prezentând anumite caracteristici comune, dar în același timp are și propriile sale proprietăți distincte [4].

Dificultatea în detecția de malware este reprezentată de imprevizibilitatea tehnicilor de pătrundere în sistem, astfel încât pentru analiza malware există trei tipuri de abordări:

Analiză statică - se analizează fișierul fără să fie executat și se examinează anumite trăsături cheie cum ar fi: antetul, metadate, șiruri de caractere, resursele și codul. Acest tip de analiză poate include și aplicarea tehnicilor de *reverse engineering* pe un fișier.

Analiză dinamică - este o tehnică care execută fișierul malițios într-un mediu simulat sau izolat și monitorizează comportamentul și efectele acestuia. Aceasta poate utiliza instrumente precum mașini virtuale și *sandbox*-uri pentru a observa comportamentul fișierului. Astfel analiza dinamică a malware-ului poate ajuta la înțelegerea scopului, modul în care funcționează și care sunt posibilele victime. De asemenea, se pot identifica IoC-uri (indicatori de compromis) ale malware-ului care oferă informații suplimentare despre fișierul respectiv.

Analiză hibridă - este o tehnică prin care un fișier este analizat dinamic prima dată iar apoi asupra rezultatelor obținute este aplicată analiza statică.

În această lucrare se propune utilizarea analizei hibride pentru a determina comportamentul unui fișier.

[2] https://www.crowdstrike.com/cybersecurity-101/malware/types-of-malware/

1.2.2. Sandbox

Un *sandbox*[3] reprezintă un mediu izolat care permite utilizatorilor să ruleze programe sau să deschidă fișiere fără a afecta sistemul sau platforma pe care rulează. În securitate cibernetică se utilizează *sandbox*-uri pentru a testa software-uri suspecte. Fără folosirea acestei tehnologii, software-ul ar putea avea acces la toate datele utilizatorului și resursele sistemului pe o rețea, punând în pericol organizația. Utilizarea unui *sandbox* pentru detectarea malware-ului oferă un strat suplimentar de protecție împotriva amenințărilor de securitate, cum ar fi atacurile *zero-day* pe care este posibil să nu le oprească, însă poate izola atacul.

Analiza dinamică și cea hibridă sunt realizate cu ajutorul unui *sandbox* care generează un raport pe baza evenimentelor desfășurate în timpul rulării unui software. Astfel, exista *sandbox*-uri specializate pentru analizarea unui malware, fiecare cu capabilitățile sale.

Multe *sandbox*-uri pot fi personalizate prin integrări cu tehnologii existente sau prin adăugarea de module proprii pentru analizarea malware-ului. Această flexibilitate permite adaptarea *sandbox*-ului la nevoile specifice ale organizației sau ale utilizatorului. Prin integrarea cu tehnologii existente, cum ar fi soluții antivirus sau sisteme de detecție a intruziunilor, se poate obține o analiză mai precisă. De asemenea, prin adăugarea de module personalizate, se poate extinde funcționalitatea *sandbox*-ului pentru a detecta și analiza tipuri specifice de *malware*.

Un dezavantaj al *sandbox*-urilor provine din cerințele de resurse pe care le implică pentru a rula în mod corespunzător. *Sandbox*-urile necesită o cantitate semnificativă de resurse, cum ar fi capacitatea de procesare, memorie și spațiu de stocare, pentru a putea crea și gestiona un mediu de testare izolat. Aceasta poate duce la o utilizare mai mare a resurselor sistemului și poate afecta performanța generală a dispozitivului sau a aplicației care rulează *sandbox*-ul.

Pe lângă resursele necesare, configurarea și administrarea *sandbox*-urilor pot fi complexe. Este important să se configureze și să se monitorizeze *sandbox*-ul în mod corespunzător pentru a asigura izolarea eficientă și securitatea mediului de testare. Un alt dezavantaj al sandbox-urilor este că nu funcționează întotdeauna. Pe măsură ce producătorii soluțiilor de securitate dezvoltă noi instrumente și strategii pentru detectarea și prevenirea atacurilor, actorii rău intenționați găsesc modalități sofisticate de a le evita, iar *sandbox*-urile nu fac excepție. Tehnicile de evitare (eng. *evasion techniques*) a *sandbox*-urilor includ amânarea executării codului malware, mascarea tipului de fișier și analizarea hardware-ului, aplicațiilor instalate, modelelor de clicuri ale mouse-ului și fișierelor deschise și salvate pentru a detecta un mediu *sandbox*. Codul malițios se va executa doar atunci când malware-ul determină că se află într-un mediu real.

Beneficiile oferite de *sandbox*-uri, precum protecția împotriva malware-ului și posibilitatea de a testa software-ul într-un mediu sigur, pot compensa dezavantajele asociate cu resursele, administrarea acestora și posibile tehnici de evitare a execuției malware-urilor. În această lucrare este propusă utilizarea *sandbox* Cuckoo, care va fi detaliat în subcapitolul 2.3.

1.2.3. Windows API

Formatul Portable Executable (PE) este un format de fișier pentru executabile, cod obiect, DLL-uri și altele, folosit în versiunile de 32 de biți și 64 de biți ale sistemelor de operare Windows. Formatul PE este o structură de date care include informațiile necesare pentru sistemul de operare Windows pentru a putea să încarce și să gestioneze codul executabil împachetat. Fișierul Executabil Portabil (PE) este împărțit în secțiuni. Fiecare secțiune furnizează informații diferite, precum antetul fișierului, numărul de DLL-uri și numărul de apeluri API. Informațiile din antetul fișierului PE pot fi modificate ușor și pot prezenta unele diferențe față de apelurile reale ale programului [5].

API-ul Windows reprezintă un set de funcții exportate de fișiere DLL dezvoltate de Microsoft și instalate pe un calculator. Prin intermediul fișierelor .h din Windows SDK, sunt definite modurile de apelare a acestor funcții. Atunci când se include un fișier .h și se realizează legătura

[3] https://www.techtarget.com/searchsecurity/definition/sandbox

cu fișierul DLL corespunzător, este posibilă utilizarea funcțiilor API Windows. Astfel, API-ul servește drept o interfață prin care diferite programe software pot interacționa între ele. În cazul Windows API există posibilitatea de a interacționa cu sistemul de operare. Prin urmare, se înțelege de ce API-ul Windows este atât de extins, oferind o gamă largă de funcționalități și facilități pentru dezvoltarea aplicațiilor.

Printre cele mai importante biblioteci pentru utilizarea API-urilor sunt [6]:

- `kernel32.dll` este o bibliotecă esențială a sistemului de operare Windows. Aceasta furnizează funcții fundamentale necesare pentru gestionarea memoriei, a proceselor și a operațiilor de intrare/ieșire

- `user32.dll` oferă funcționalitățile de bază pentru gestionarea desktopului, ferestrelor și meniurilor. Aceasta oferă acces la API-urile necesare pentru crearea interfețelor standard cu utilizatorul

Chiar dacă apelurile API pot reprezenta o provocare în ceea ce privește utilizarea lor de către programatori, acestea ajută la realizarea multor funcționalități dorite. Cu toate că există numeroase utilizări legitime ale acestor apeluri, există și cazuri opuse. Autorii de malware folosesc aceste apeluri API pentru a interacționa cu sistemul de operare și a realiza acțiuni nelegitime [7].

În cel de-a doua situație, există apeluri API care sunt comune pentru dezvoltarea unui malware, de exemplu injectarea DLL-ului este o tactică utilizată în cadrul atacurilor cibernetice pentru a executa un DLL arbitrar în interiorul unui alt proces. Procesul de injectare implică mai multe etape din perspectiva unui atacator:

1. localizarea procesului în care dorește să injecteze DLL-ul malițios. Aceasta se realizează prin utilizarea funcțiilor `CreateToolhelp32Snapshot`, `Process32First` și `Process32Next` pentru a obține informații despre procesele aflate în execuție.

2. deschiderea procesului țintă utilizând funcțiile `GetModuleHandle`, `GetProcAddress` și `OpenProcess`. Acestea îi permit să obțină o cale (eng. *handle*) către procesul respectiv, oferind acces la memoria acestuia.

3. scrierea căii către DLL în interiorul memoriei procesului. Acest lucru se realizează prin utilizarea funcțiilor `VirtualAllocEx` și `WriteProcessMemory` pentru a alocate spațiu în memoria procesului și a scrie conținutul DLL-ului în acea zonă de memorie.

4. crearea unui fir de execuție în cadrul procesului, care va încărca DLL-ul malițios utilizând funcțiile `CreateRemoteThread` și `LoadLibrary`.

Prin această tehnică[4] de injectare a DLL-ului, atacatorii pot obține controlul asupra procesului țintă și pot executa codul lor malițios în cadrul acestuia, având potențialul de a cauza daune semnificative și de a compromite securitatea sistemului.

De exemplu autorii articolului [8] consideră că pentru atacuri de tip DLL injection următoarea secvență de api-uri este critică `OpenProcess`, `VirtualAllocEx`, `WriteProcessMemory`, `CreateRemoteThread`. Se observă că API-urile din secvență se regăsesc în etapele anterior.

În [6] a fost utilizată o metodă de extragere a caracteristicilor apelurilor API și de antrenare a clasificatorilor folosind mai multe tehnici de *data mining*, în vederea unei detectări eficiente a malware-ului obfuscat care conduc la atacuri *zero-day* (necunoscute) de astăzi. Prin citarea acestui articol, am intenționat să subliniem faptul că există posibilitatea identificării unor atacuri de tip *zero-day* prin analizarea apelurilor Windows API.

[4] https://book.hacktricks.xyz/reversing-and-exploiting/common-api-used-in-malware

Utilizarea apelurilor API pentru detectarea malware-ului a reprezentat o preocupare a comunității INFOSEC[5] din aceeași perioadă, fiind analizate pe parcursul etapei de documentare și articolele [9], [10] și [11]. În această lucrare de diplomă, am adoptat aceeași abordare, folosind apelurile API ca principală caracteristică pentru analiza și detectarea malware-ului.

1.2.4. Reguli Yara

Regulile Yara reprezintă o formă de limbaj de programare, care funcționează prin definirea unui număr de variabile ce conțin modele găsite într-o colectie de malware. Dacă unele sau toate condițiile sunt îndeplinite, în funcție de regulă, atunci acestea pot fi utilizate pentru a identifica cu succes o parte dintr-un malware. Un avantaj al regulilor Yara este reprezentata de simplitatea și ușurință cu care se creează [12]. În Listing 1.1 este prezentată o regulă simplă care se potrivește cu orice string care conține textul "foobar", în cazul în care este necesară o condiție mai complicată ele pot conține expresii regulate sau alte modalități de potrivire în funcție de nevoile utilizatorului.

```
rule TextExample
{
  strings:
    $text_string = "foobar"

  condition:
    $text_string
}
```

Listing 1.1. Exemplu de regulă Yara pentru detectarea cuvântului *foobar*, conform documentației Yara[6]

Astfel, regulile Yara reprezintă o modalitate de creare a tiparelor personalizate care este utilă pentru mecanismul de partajare sau detectare a informațiilor despre malware. [13]

1.2.5. Corelarea datelor într-o platformă de tip Threat Intelligence

Există mai multe platforme de Threat Intelligence, iar printre cele mai cunoscute se numără OpenCTI, CrowdStrike Falcon X, Anomali și altele. Cu toate acestea, MISP se evidențiază în fața competiției prin vizualizarea simplă a datelor și poate fi potrivită pentru persoane care nu sunt neapărat specializate în domeniu. De asemenea, MISP beneficiază de o comunitate activă, este gratuit și oferă integrări facile cu alte soluții. Aceste caracteristici fac din MISP o alegere potrivită pentru această lucrare.

Integrarea soluției cu platforma MISP permite partajarea eficientă a rapoartelor, informațiilor și indicatorilor de compromitere între diferite entități, cum ar fi organizații și cercetători în domeniul securității cibernetice. Această integrare facilitează schimbul de cunoștințe și experiențe în ceea ce privește amenințările cibernetice, contribuind la o înțelegere mai amplă și completă a peisajului de securitate. De asemenea, MISP oferă funcționalități avansate de analiză și corelare a datelor, permițând identificarea automată a relațiilor și pattern-urilor între diversele atribute și indicatori proveniți de la malware, atacuri și campanii de analiză.

1.2.6. Învățare automată

Învățarea automată este o ramură a inteligenței artificiale (IA) și informaticii care se concentrează pe utilizarea datelor și algoritmilor pentru a imita modul în care oamenii învață, îmbunătățind treptat precizia acestora. Această învățare este de două tipuri, supervizată și nesupervizată.

Învățarea supervizată este utilizată în contexte în care datele de antrenare și de testare aparțin unor anumite clase cunoscute în avans, având ca scop încadrarea noilor instanțe în acea

[5]Information Security

[6]https://yara.readthedocs.io/en/v3.4.0/writingrules.html

clasă. Aceasta se împarte în algoritmi de regresie, care încearcă să estimeze o valoare pentru o nouă instanță, și algoritmi de clasificare, care sunt potriviți, de exemplu, pentru clasificarea fișierelor de tip *malware* unde cele două categorii sunt reprezentate de software rău intenționat și software inofensiv.

Învățarea nesupervizată, utilizează algoritmi de învățare automată pentru a analiza și grupa seturi de date care nu au o anumita clasă de care aparțin de la început. Acești algoritmi descoperă modele ascunse sau grupări de date fără a avea nevoie de intervenția umană. Un exemplu de utilizare pentru învățarea nesupervizată este atunci când avem un set de imagini care conține atât pisici, cât și câini, iar dorim să grupăm aceste imagini în cele două categorii fără a avea informații prealabile despre conținutul fiecărei imagini. Algoritmul de învățare nesupervizată va analiza caracteristicile și modelele din setul de imagini și va găsi similarități între ele, bazându-se pe atribute cum ar fi forma, textura sau culorile predominante. Pe baza acestor similarități, algoritmul va grupa automat imaginile în două categorii separate, în acest caz, pisici și câini. Astfel, algoritmul poate identifica și separa imaginile care reprezintă pisici de cele care reprezintă câini, fără a necesita intervenția umană pentru etichetarea manuală a fiecărei imagini.

În domeniul securității cibernetice învățarea automată poate fi utilizată pentru detectare de malware, automatizare de task-uri și alte cazuri prezentate în Figura 1.2. În această lucrare se utilizează un algoritm de învățare automată pentru a clasifica un fișier în malware sau goodware.

Use Case	Description
Vulnerability Management	Provides recommended vulnerability prioritization based on criticality for IT and security teams
Static File Analysis	Enables threat prevention by predicting file maliciousness based on a file's features
Behavioral Analysis	Analyzes adversary behavior at runtime to model and predict attack patterns across the cyber kill chain
Static & Behavioral Hybrid Analysis	Composes static file analysis and behavioral analysis to provide advanced threat detection
Anomaly Detection	Identifies anomalies in data to inform risk scoring and to direct threat investigations
Forensic Analysis	Runs counterintelligence to analyze attack progression and identify system vulnerabilities
Sandbox Malware Analysis	Analyzes code samples in isolated, safe environments to identify and classify malicious behavior, as well as map them to known adversaries

Figura 1.2. Utilizare de tehinici ML in securitate[7].

Ansambluri de arbori de decizie (Random Forest) este un algoritm de învățare supervizată care se bazează pe algoritmul de învățare automată: arbore de decizie. Structura unui Random Forest este prezentată în Figura 1.3. Arborele de decizie este compus dintr-un nod rădăcină, noduri intermediare, noduri frunză și muchii. Nodul rădăcină reprezintă caracteristica cea mai semnificativă care ajută la clasificarea sau predicția datelor. Nodurile intermediare reprezintă deciziile luate pe baza valorilor atributelor în ordinea descendentă a importanței lor decizionale, nodurile frunză reprezintă rezultatele finale, adică etichetele claselor sau valorile prezise, iar muchiile reprezintă valorile (sau grupurile de valori ale) atributelor [14]. Pentru a alege atributele potrivite pentru fiecare nod există mai multe tehnici însă cele mai populare sunt impuritatea Gini și câștigul informațional (IG – Information Gain) care se determină ca fiind măsura în care se reduce entropia mulțimii datelor de antrenare.

[7]imagine preluată de pe urmatorul site web: https://www.crowdstrike.com/cybersecurity-101/machine-learning-cybersecurity/

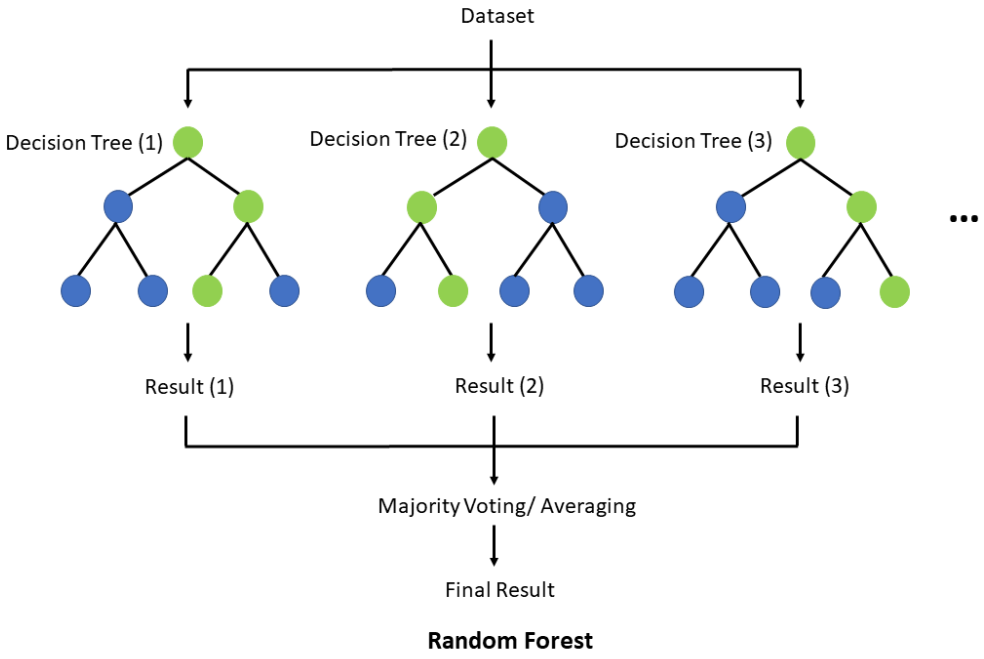

Figura 1.3. Structura unui Random Forest[8].

Random Forest este o metodă de învățare bazată pe ansambluri, aceste modele funcționează pe principiul conform căruia un grup de „clasificatori slabi" (*weak classifiers*) pot forma, împreună, un clasificator puternic (*strong classifier*). Un arbore singular este susceptibil la fenomenul de *overfitting*, astfel Random Forest evită problema aceasta prin utilizarea mai multor arbori de decizie.

Clasa finala de care apartine instanța nouă este determinată prin clasa care apare cel mai frecvent printre mulțimea de clase furnizată de arbori.

În aceasta lucrare s-a folosit algoritmul de Random Forest pentru a clasifica un fișier în clasa de malware sau goodware.

1.3. Referințe la subiecte similare

Acest subcapitol este împărțit în două subsecțiuni. Prima secțiune se concentrează pe compararea și evaluarea *sandbox*-urilor existente. Astfel, sunt analizate avantajele și dezavantajele diferitelor soluții disponibile, precum și caracteristicile și funcționalitățile specifice. Sunt evidențiate aspecte precum nivelul de flexibilitate, capacitatea de analiză, suportul pentru module suplimentare și alte caracteristici relevante.

A doua secțiune se axează pe soluțiile care implică integrarea unor module personalizate cu un *sandbox* sau soluții prin care se analizează fișiere pentru a determina comportamentul acestora. Aceasta aspect presupune analizarea modalităților prin care module propii pot fi integrate într-un *sandbox* pentru a îmbunătăți analiza de securitate. De asemenea, se vor explora exemple și studii de caz în care module personalizate au fost utilizate pentru a extinde funcționalitatea unui *sandbox* existent, aducând beneficii suplimentare în procesul de analiză.

Prin explorarea și compararea acestor aspecte, se urmărește identificarea celor mai potrivite soluții și abordări în utilizarea *sandbox*-urilor și integrarea modulelor pentru a obține rezultate precise în analiză.

[8]imagine preluată de pe următorul site web: https://en.m.wikipedia.org/wiki/File:Random_forest_explain.png

1.3.1. Tehnologii de sandboxing

1.3.1.1. Cuckoo

Lucrarea utilizează *sandbox*-ul "Cuckoo" pentru implementarea soluției propuse. Acesta este un instrument puternic care poate analiza automat orice fișier potențial malware sub diferite sisteme de operare, inclusiv Windows, macOS, Linux și Android. Prin simpla trimitere a unui fișier suspect către Cuckoo, software-ul va executa fișierul într-un mediu controlat și izolat și va oferi un raport detaliat asupra comportamentului fișierului [15].

Cuckoo este o unealtă gratuită și puternică care ajută prevenirea amenințărilor prin automatizarea procesului de analiză malware. Prin Cuckoo se obțin rapid și ușor informații cu privire la comportamentul fișierelor potențial malware. Această abordare permite utilizatorilor sa ia decizii informate în ceea ce privește modul de gestionare și reacționare la amenințările detectate.

Unul dintre avantajele oferite de Cuckoo este faptul că vine cu o interfață ușor de utilizat, care prezintă în mod clar și accesibil rezultatele obținute în urma analizei fișierelor suspecte. Prin intermediul acestei interfețe, utilizatorii pot accesa toate analizele procesate de Cuckoo, care sunt:

- Rezumatul analizei - care oferă informații generale despre fișierul analizat și rezultatele obținute în urma executării sale în mediu izolat.
- Analiza dinamică - care prezintă informații detaliate despre comportamentul fișierului în timpul execuției în mediul izolat, inclusiv apeluri de sistem, procese, comunicare de rețea și alte activități suspecte.
- Analiza statică - care examinează fișierul din punct de vedere al semnăturilor și a altor atribute statice pentru a detecta semne de malware.
- Capturi de ecran - care prezintă capturi de ecran realizate în timpul execuției fișierului suspect în mediul izolat, pentru a oferi o imagine mai clară a modului în care acesta acționează.
- Raportul de rețea - care oferă informații detaliate despre activitatea de rețea a fișierului suspect, inclusiv adrese IP, porturi, protocoale și alte informații relevante.
- Module opționale suportate de Cuckoo - acestea sunt module pentru care Cuckoo a facilitat integrarea, un exemplu ar fi MISP.

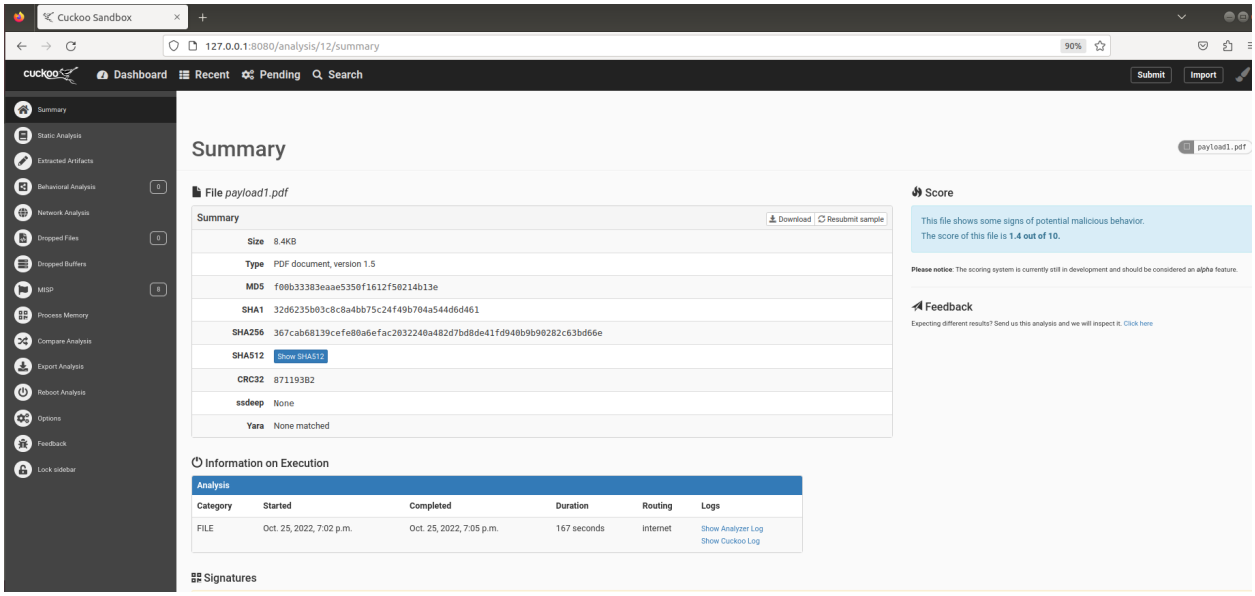

Figura 1.4. Interfață Cuckoo pentru vizualizarea unui fișier analizat.

Toate aceste analize pot fi accesate și vizualizate într-un mod intuitiv și ușor de înțeles prin intermediul interfeței Cuckoo (Figura 1.4), oferind astfel utilizatorilor posibilitatea de a evalua rapid și eficient potențialele amenințări și de a lua măsuri de protecție adecvate.

Pentru o analiza mai amănunțită Cuckoo oferă un raport detaliat în format JSON care conține informații despre comportamentul fișierului analizat în mediul izolat. Acest raport este foarte util pentru a analiza comportamentul unui malware-ului și poate include informații precum:

- Activități de rețea: adrese IP, porturi, protocoale, pachete transmise și primite etc.
- Activități de sistem: procese, servicii, module încărcate, apeluri API etc.
- Activități de fișiere: fișiere create, citite, scrise, șterse, modificate etc.
- Activități de regiștri: chei de regiștri create, citite, scrise, șterse, modificate etc.
- Alte activități: interacțiune cu driveri, manipularea ferestrelor și a tastaturii, comunicare cu alte procese, crearea de procese ascunse etc.

Acest raport cu structura din Figura 1.5 poate fi folosit pentru a identifica și analiza comportamentul specific al malware-ului și poate fi folosit pentru a dezvolta alte metode de detectare și prevenire a atacurilor similare.

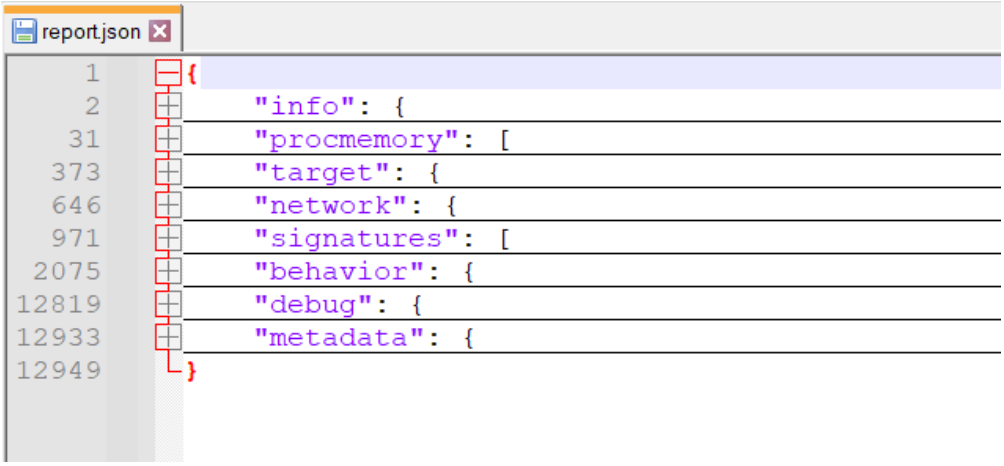

Figura 1.5. Structură generală a unui raport generat de Cuckoo.

Cuckoo are un API ușor de utilizat și de extins, care permite utilizatorilor să integreze *sandbox*-ul în fluxul lor de lucru existent și să extindă funcționalitatea acestuia prin dezvoltarea modulelor suplimentare.

API-ul Cuckoo oferă o interfață RESTful prin care se pot trimite solicitări HTTP pentru a controla *sandbox*-ul și a accesa informațiile despre analizele efectuate. Acest API poate fi folosit pentru a automatiza analizele malware-ului, pentru a integra *sandbox*-ul în alte instrumente de securitate sau pentru a dezvolta module suplimentare pentru Cuckoo.

De exemplu, un utilizator poate dezvolta un modul suplimentar pentru Cuckoo care să utilizeze o tehnologie de analizare a API-urilor.

1.3.1.2. ANY.RUN

ANY.RUN este o platformă bazată pe tehnologia cloud care permite utilizatorilor să încarce și să analizeze fișiere și URL-uri suspecte. Furnizează o interfață prietenoasă cu utilizatorul ca în Figura 1.6, și o gamă largă de instrumente de analiză, inclusiv analiză dinamică, analiză statică și analiză comportamentală [16].

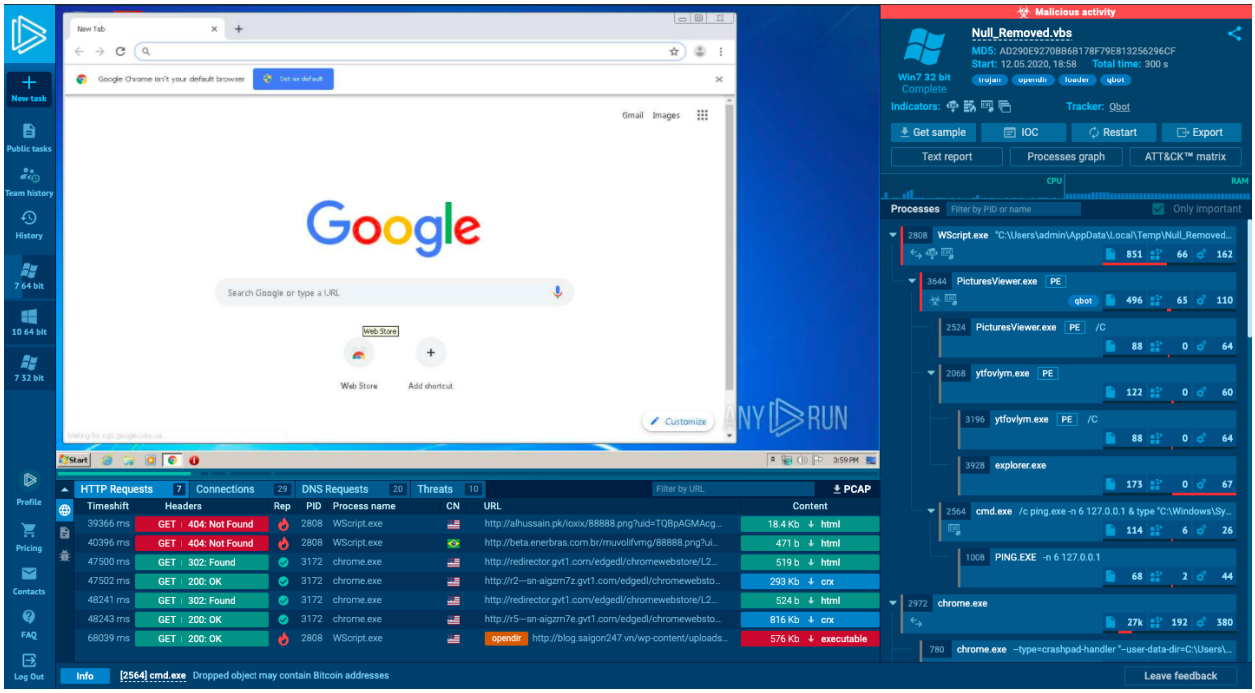

Figura 1.6. Interfața sandbox-ului ANY.RUN.

ANY.RUN oferă suport *live* prin intermediul *chat*-ului online, care poate fi accesat prin pagina de contact a site-ului. Utilizatorii pot accesa chat-ul online pentru a obține asistență tehnică sau pentru a solicita ajutor cu problemele legate de utilizarea platformei ANY.RUN.

Echipa de asistență tehnică ANY.RUN este formată din experți în securitate, care sunt pregătiți să răspundă la întrebările și solicitările utilizatorilor și să ofere soluții la problemele tehnice legate de platformă.

Un dezavantaj al *sandbox*-ului ANY.RUN este faptul că este suportat doar pe sistemele de operare Windows și că deși există o versiune gratuită, nu toate funcțiile și capacitățile sunt disponibile.

Pentru utilizatorii care folosesc alte sisteme de operare, cum ar fi macOS sau Linux, acesta poate fi un dezavantaj major, deoarece nu pot utiliza ANY.RUN pentru analiza fișierelor și a URL-urilor lor.

Alegerea dintre ANY.RUN și Cuckoo depinde de nevoile specifice ale utilizatorului. Dacă utilizatorul preferă o soluție bazată pe cloud și suport *live* ANY.RUN poate fi cea mai bună opțiune. Cu toate acestea, dacă utilizatorul are nevoie de o platformă *sandbox* foarte flexibilă și extensibilă care poate fi integrată cu diverse instrumente și servicii, Cuckoo este alegerea mai bună.

1.3.1.3. FortiSandbox

FortiSandbox este o soluție de securitate avansată bazată pe *sandbox*-ing care ajută organizațiile să detecteze și să prevină atacurile cibernetice prin analiza automată a fișierelor malware și a amenințărilor necunoscute. Acesta oferă capacități de analiză dinamică și de emulare a sistemului de operare, folosind tehnici de învățare automată prin care sunt identificate și izolate malware-uri [17] care pot să pătrundă într-un sistem de securitate.

FortiSandbox poate fi integrat cu soluțiile de securitate existente ale organizației și poate analiza diverse tipuri de amenințări, inclusiv fișiere executabile, fișiere de pe web, macro-uri și atașamente de e-mail. Soluția utilizează tehnologia de analiză în timp real pentru a identifica și a bloca amenințările noi și necunoscute.

De asemenea, FortiSandbox oferă capacități de analiză a rețelei, permițând identificarea amenințărilor care folosesc tehnici de atac sofisticate pentru a pătrunde într-un sistem. Soluția poate să identifice atacurile de tipul *zero-day* și poate să ofere informații detaliate despre sursă.

FortiSandbox poate fi implementat ca o soluție hardware sau software și este disponibil în diferite planuri de abonament, cu diferite capacități și funcționalități. Soluția poate fi personalizată în funcție de nevoile organizației și poate fi integrată cu alte instrumente de securitate Fortinet ca în Figura 1.7.

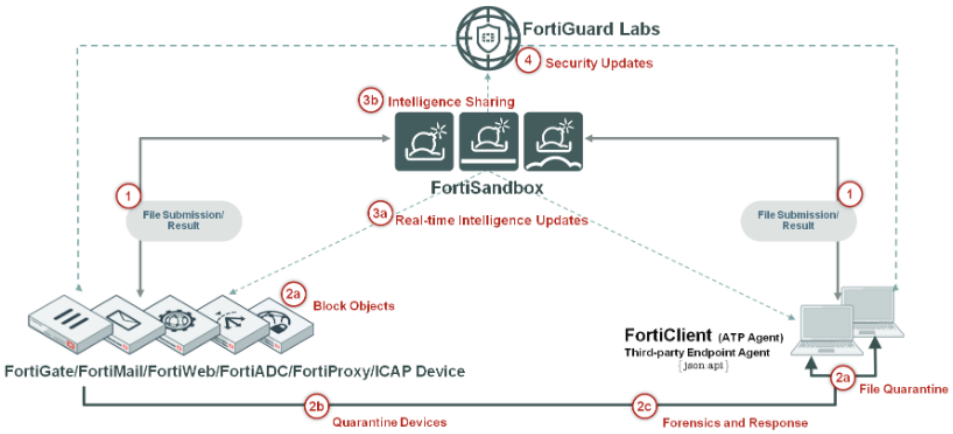

Figura 1.7. Fluxul de lucru pentru gestionarea amenințărilor în FortiSandbox[9].

Una dintre principalele diferențe între cele două soluții este faptul că FortiSandbox este o soluție comercială, în timp ce Cuckoo este o soluție *open-source*. FortiSandbox este oferit ca un produs hardware sau software, în timp ce Cuckoo necesită ca utilizatorii să configureze și să ruleze *sandbox*-ul pe propriile lor sisteme.

O altă diferență importantă între cele două soluții este nivelul de personalizare și flexibilitate. Cuckoo este o soluție extrem de flexibilă, permițând utilizatorilor să configureze *sandbox*-ul în funcție de nevoile lor specifice. În schimb, FortiSandbox oferă mai puține opțiuni de personalizare, dar este mai ușor de configurat și de implementat.

În ceea ce privește capacitățile de detecție a malware-ului, ambele soluții utilizează tehnologii de analiză dinamică și de emulare a sistemului de operare pentru a identifica amenințările.

De asemenea o alta deosebire este că FortiSandbox nu dispune de un API deschis și extensibil, ceea ce limitează posibilitățile de integrare cu alte instrumente și servicii. Cu toate acestea, Fortinet oferă un set de API-uri pentru a permite integrarea cu produsele proprii Fortinet.

1.3.2. Soluții similare de malware analysis

Există două tipuri de analiză a malware-ului: analiza statică și analiza dinamică. Analiza statică constă în examinarea fișierului fără a-l executa, în timp ce analiza dinamică implică rularea malware-ului într-un mediu controlat [18].

Deoarece în cadrul analizei statice, binarul unui malware este procesat fără a rula codul, acest tip de analiză se concentrează pe identificarea semnăturilor fișierului sau pe utilizarea unor tehnici de *reverse engineering*, în care codul este analizat pentru a înțelege structura și scopul acestuia.

Prin analiza dinamică, execuția fiind făcută într-un mediu controlat și izolat, este evitat riscul de infectare a sistemului gazdă. Această analiză permite observarea detaliată a comportamentului malware-ului, identificarea activităților suspecte și a interacțiunilor cu alte aplicații sau sistemul de operare. Prin urmărirea fluxului de date și a interacțiunilor, se poate determina scopul și intențiile malware-ului, precum și eventualele vulnerabilități pe care le exploatează.

Prin integrarea analizei statice și dinamice, se poate realiza o evaluare mai precisă a malware-ului. Această abordare combinată oferă posibilitatea de a efectua o analiză statică a datelor gene-

[9] https://www.fortinet.com/content/dam/fortinet/assets/data-sheets/FortiSandbox.pdf

rate prin analiza comportamentală, extinzând astfel capacitatea de detectare a amenințărilor de tip *zero-day* și evaluare a semnelor de atac.

1.3.2.1. Detecție de malware prin creare de reguli bazate pe mai multe caracteristici

Articolul "Combined dynamic multi-feature and rule-based behavior for accurate malware detection" abordează o problema veche însa importantă din securitate, anume detectarea de malware-uri. Malware-ul reprezinta un software rău intenționat care poate fi utilizat în diverse scopuri ilegale, cum ar fi obținerea de acces neautorizat la date.

În general malware-ul poate fi detectat prin doua metode principale, prin semnături sau/și prin comportament. Detectarea bazată pe semnătură utilizează semnături de malware existente pentru a detecta atacurile cunoscute, iar detectarea bazată pe comportament se concentrează pe analiza comportamentului malware-ului [19].

Articolul propune o abordare noua pentru detecția bazată pe comportament cu arhitectura din Figura 1.8. Mai exact, abordarea propusă în articol utilizează analiza dinamică pentru a identifica comportamentele suspecte ale malware-ului, iar apoi utilizează analiza de comportament bazată pe reguli pentru a confirma dacă comportamentul este malițios sau nu.

Autorii afirmă că există mai multe modalități de a detecta un comportament suspect al malware-ului, însă consideră că cele mai importante informații referitoare la modul în care malware-ul se desfășoară pot fi extrase din următoarele trei elemente:

- API calls/ Apeluri de API, acestea sunt reprezentate de apeluri făcute de sistemul de operare către alte aplicații/programe, și sunt esențiale pentru interacțiunea dintre acestea. Un exemplu concret de astfel de apeluri este funcția "NtOpenFile", care este folosită pentru a deschide fișiere sau directoare în sistemul de operare Windows.

- API sequence / Secvente API, acestea sunt alcătuite din apelurile de sistem făcute de un proces și reprezintă un șir de instrucțiuni care descriu interacțiunea dintre un program și sistemul de operare. Aceste secvențe sunt importante pentru analiza comportamentului unui proces sau program și pot fi folosite pentru a detecta activități suspecte sau potențial dăunătoare.

- Trafic de internet care reprezintă fluxul de date care circulă între un dispozitiv și serverele conectate la internet, acesta poate fi analizat pentru a observa cererile făcute de către malware. Prin monitorizarea traficului de internet, se pot detecta activități suspecte sau potențial dăunătoare ale programelor malware, cum ar fi descărcarea altor fișiere malware sau comunicarea cu servere de C&C (*Command and Control*) .

Astfel, prin elementele enumerate mai sus, se poate determina comportamentul unui fișier executat și pentru extragerea acestor informații autorii propun utilizarea *sandbox*-ului Cuckoo.

Din rezultatele preluate din raportul generat de *sandbox*, se aplica doi algoritmi (TF-IDF si LCS) pentru extragerea de API-uri care ar putea sa determine comportamentul fișierului. În urma aplicării lor, se pot genera reguli Yara care determina dacă intențiile fișierului sunt dăunătoare sau nu.

Aceasta reprezintă etapa de antrenare pentru a crea reguli. În etapa de testare sunt extrase doar informațiile necesare din raportul generat de *sandbox*, iar apoi aceste date sunt potrivite cu regulile YARA, urmând pe baza unui sistem de votare sa se decidă dacă fișierul analizat are sau nu intenții rele. Acest proces de votare ia în considerare toate regulile aplicabile și decide în funcție de numărul de voturi pozitive sau negative pentru fiecare regulă. În acest fel, se poate identifica dacă fișierul executabil este un program malware sau nu.

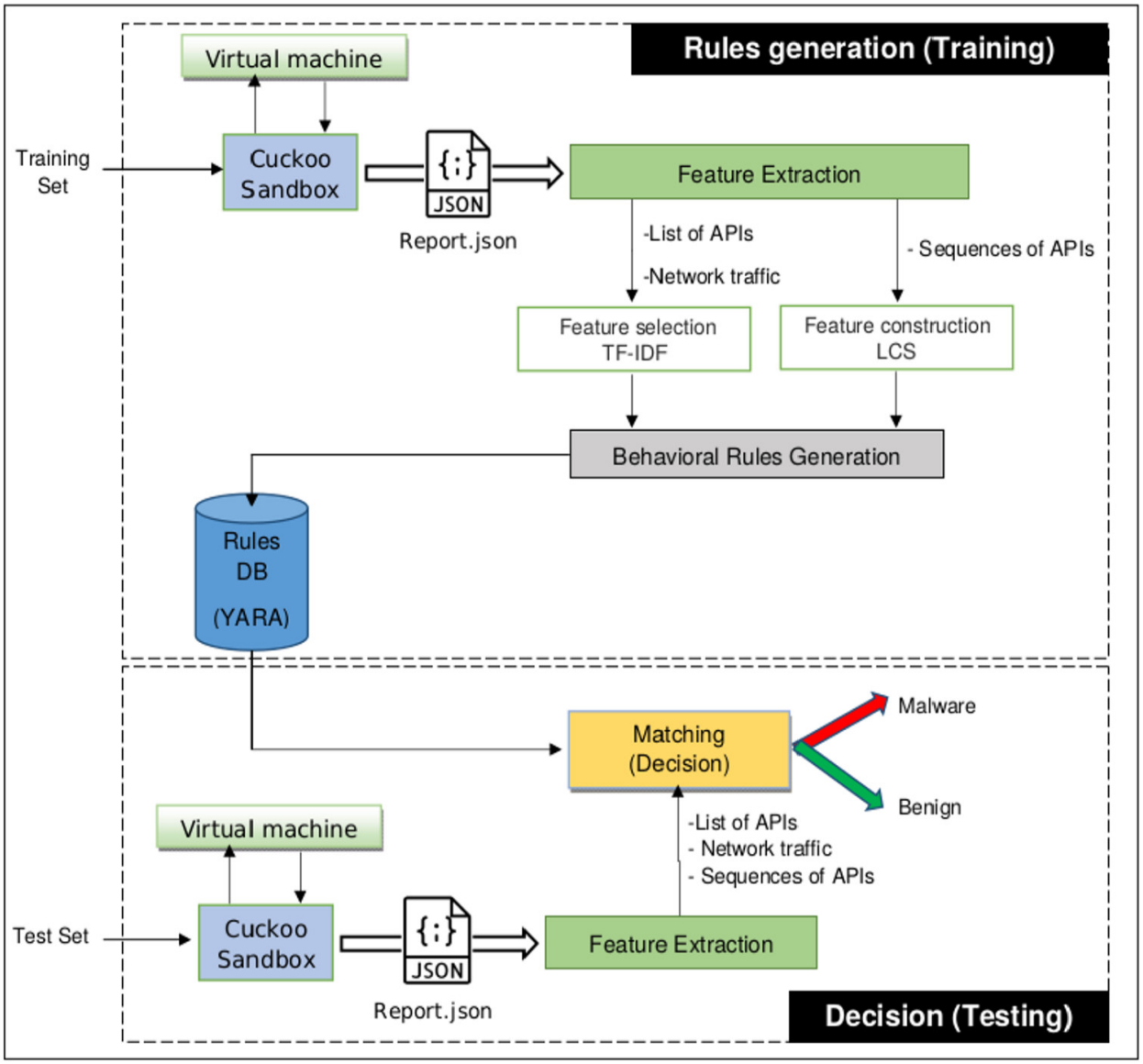

Figura 1.8. Arhitectură propusă în articolul [19].

Pentru a evalua eficacitatea abordării propuse, autorii au efectuat experimente pe un set de date care conținea diverse tipuri de malware. Rezultatele experimentelor au arătat că abordarea propusă a obținut o precizie ridicată în detectarea malware-ului și are o performanță mai bună decât alte abordări existente.

1.3.2.2. Detecție de malware bazată pe apeluri API și analiză comportamentală

Alazab et al.18 [20] au prezentat o abordare de detectare a malware-ului bazată pe comportament folosind apelurile API ca trăsături comportamentale, care sunt extrase automat și apoi reprezentate ca n-grame. Pentru sarcina de clasificare, autorii au folosit un algoritm de învățare automată (SVM). Aceștia au prezentat rezultate foarte promițătoare cu o acuratețe de 96,50% folosind reprezentările n-gram de dimensiune unu.

Pentru a ajunge la aceste rezultate autorii au implementat arhitectura din Figura 1.9, se observă ca aceștia nu au folosit un *sandbox* pentru extragerea apelurilor API ci mai întâi au decompresat fișierul iar apoi utilizând IDA Pro au extras apelurile API care sunt ulterior prelucrate și clasificate. Partea mai relevantă a studiului constă în modul în care aceste apeluri API sunt interpretate și aplicate pe un model de *machine learning* pentru a ajunge la un rezultat cat mai precis.

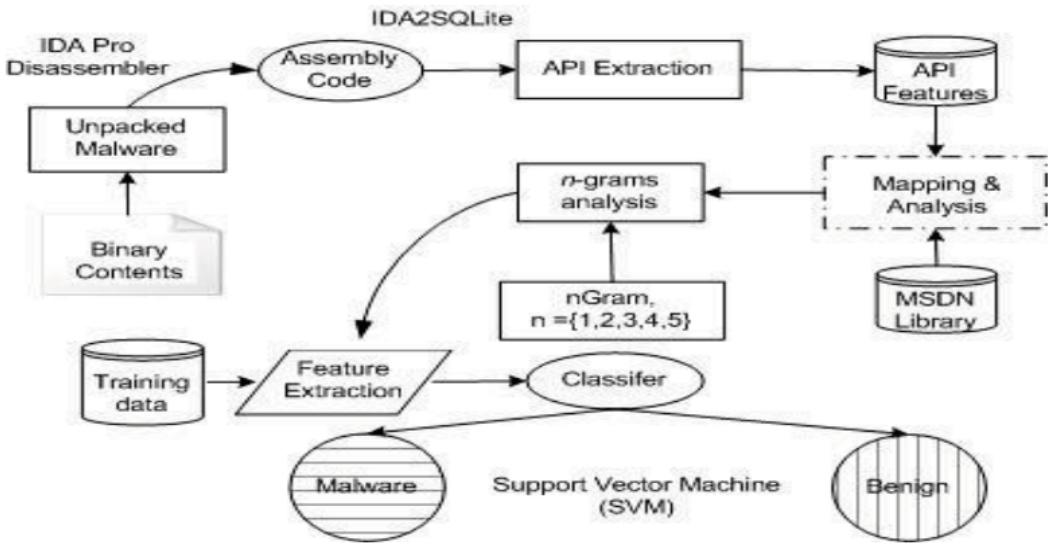

Figura 1.9. Arhitectura propusă în articolul [20].

Modalitatea prin care este detectat malware în această lucrare se desfășoară în cinci etape:

1. S-a decompresat malware-ul și s-a desasamblat fișierul executabil pentru a obține programul în limbaj de asamblare.
2. S-au extras apelurile API din programul în limbaj de asamblare.
3. S-au asociat apelurile API cu biblioteca MSDN și s-au analizat.
4. S-au extras caracteristicile N-gram.
5. S-a antrenat un clasificator folosind mașina cu vectori de suport (SVM) și s-a construit un model.

Pentru evaluarea soluției au fost utilizate în total 314 fișiere executabile, dintre care 72 sunt fișiere inofensive și 242 sunt fișiere malware, s-a observat că pentru o dimensiune mai mare a n-gram-ului scade precizia modelului.

1.3.2.3. Clasificare de malware bazată pe apeluri API și analiză comportamentală

Pektaş și Acarman [21] propun o metodă de clasificare a malware-ului bazată pe integrarea multiplă a caracteristicilor dinamice. Aceștia extrag caracteristicile de comportament ale malware-ului, precum activitățile care se execută prin intermediul sistemului de fișiere, rețea, *Windows Registry* observate în urma urmăririi execuției și modelarea n-gramelor peste secvențele de apeluri API. Prin aceasta modelare se vor extrage caracteristici de comportament care vor fi prelucrate printr-un sistem de votare pentru a extrage modele/secvente suspecte de apeluri API. Modelul de clasificare este construit prin aplicarea algoritmilor de învățare automată online și este comparat cu clasificatorii de bază pentru a evalua performanța sa.

Autorii utilizează VirMon pentru a extrage trăsăturile care țin de fișiere, registre, procese, activități de rețea și alertele sistemului de detecție a intruziunilor (IDS) iar Cuckoo este utilizat pentru a extrage secventele API. Pentru partea de extragere de caracteristici din secventa de API-uri, acestea au fost codificate și asociate unei categorii, de exemplu apelul unhookwindowshookex este în categoria de Hooking și se codifica prin "Aa". Ulterior toate caracteristicele sunt asociate

unei categorii, de exemplu categoria de secvente api conține api-urile codificare anterior peste care a fost aplicat n-gram și s-a obținut un rezultat precum: 'FbFcDaFc'.

Determinarea secventelor API relevante se face prin crearea unui arbore pentru mai multe secvente, fiecare nod conține frecvența unui api și entropia sa, iar fii reprezinta api-uri care fost apelate imediat după nodul părinte. Pe aceste valori se aplica n-gram pentru a vedea care sunt secventele relevante în funcție de entropie și nodul părinte.

Pentru clasificare au fost utilizați algoritmii SVM, RF, K-NN și CW [22] pe un set de date de 17 400 de malware-uri și 532 de executabile Windows. Cea mai mare acuratețe s-a obținut utilizând algoritmul CW cu o valoarea de 98%.

1.3.2.4. Studiu de detecție și clasificare de malware bazată pe apeluri API utilizând rețele neuronale recurente și algoritmi tradiționali de învățare automată

Cannarile [23] a propune o analiză a diferitelor tehnici pentru detectarea malware-ului și ulterior clasificarea acestora prin identificarea familiei din care fac parte. Pentru a realiza acest lucru, algoritmi de învățare automată bazată pe arbori sunt comparați cu algoritmi de învățare adâncă bazată pe rețele neurale recurente.

Pentru realizarea acestei lucrări au fost utilizați următorii algoritmi: Random Forest, CatBoost, XGBoost, ExtraTrees, TabNet, Bi-LSTM, Bi-GRU, iar ca set de date pentru detectare s-a folosit malware-analysis-datasets-api-call-sequences și APIMDS. Pentru partea de clasificare în familii Catak si APIMDS. Astfel pentru setul de date malware-analysis-datasets-api-call-sequences pentru detecție de malware a obținut cea mai buna precizie pentru Bi-GRU (90%), iar pentru setul de date APIDMS cel mai bun algoritm este tot Bi-GRU (99%), este de menționat faptul ca algoritmii bazați pe arbori tot au o precizie buna de aproximativ 98%. Pentru partea de clasificare pentru setul de date APIDMS cel bun rezultat este obținut prin Ranfom Forest (91.1%) și pentru Catak este ExtraTrees (59.3 %).

În experimentul pentru detectarea malware-ului, se poate observa că algoritmii de învățare adâncă nu obțin întotdeauna performanțe mai bune decât algoritmii de învățare automată și viceversa. Este de remarcat faptul că, pentru ambele seturi de date analizate în cadrul detectării malware-ului, algoritmii Bi-LSTM și Bi-GRU (care sunt folosiți în general pentru clasificarea secvențelor textuale) obțin întotdeauna performanțe mai ridicate decât algoritmii bazați pe arbori, însa nu este o diferență substanțială.

1.3.2.5. Clasificare de malware bazată pe vizualizarea secvențelor API

Qian și Tang [24] propun o altă metodă de detecție a malware-ului prin analiza atributelor API și împărțirea lor în 16 categorii. Autorii propun o metodă de colorare a secvențelor API pe baza categoriilor și a frecvenței de apariție a API-urilor în unitatea de timp în care acestea sunt executate. Într-o etapă ulterioară, autorii propun utilizarea unui model de rețele neuronale convoluțiunile (CNN) pentru a construi un clasificator.

Deși metoda propusă de Qian și Tang se remarcă prin precizia ridicată și capacitatea de a detecta cu succes diverse tipuri de malware, considerăm că prezintă dezavantajul unei cantități semnificative de date de antrenament pentru a produce rezultate precise, ceea ce poate fi o problemă deoarece aceste date, în cele multe cazuri, sunt limitate.

Mai multe abordări identificate în urma analizei efectuate au demonstrat utilitatea combinării analizei atributelor API prin intermediul *sandbox* cu o altă soluție care diferă de la autori la autori. Lucrarea propune combinarea mai multor soluții existente într-un mod modular, cu scopul de a îmbunătăți acuratețea analizei prin îmbunătățirea precizia regulilor Yara, utilizând module de clasificare a API-urilor.

1.4. Metoda propusă

În lucrare se urmărește identificarea malware-urilor prin intermediul analizei comportamentale bazate pe integrarea modulară a mai multor soluții existente. Comportamentul malware-urilor este determinat prin analizarea rapoartelor *sandbox*, unde sunt identificate cele mai semnificative atribute prin apelurile de sistem și trafic de internet. Metoda propusă implică generarea regulilor Yara și antrenarea unui model de învățare automată pe baza unor atribute extrase din raportul generat de *sandbox*. Pentru generarea regulilor Yara, se extrag trei atribute cheie: traficul de internet, cele mai importante apeluri de sistem și cea mai lungă secvență comună de apeluri de sistem a unui fișier. Aceste atribute sunt obținute prin aplicarea algoritmilor TF-IDF și LCS asupra raportului. Pentru antrenarea modelului de învățare automată, se selectează doar anumite atribute relevante referitoare la apelurile API. După finalizarea fazei de antrenare, pentru a determina dacă un fișier este malițios sau nu, se extrag atributele necesare pentru cele două module: modulul de reguli Yara și modulul de învățare automată. În cadrul modulului de reguli Yara, cele trei atribute ale fișierului sunt comparate cu regulile Yara generate anterior, iar o decizie este luată prin votare. Dacă fișierul se potrivește cu mai multe reguli generate de fișiere inofensive, atunci este considerat sigur. În caz contrar, dacă se potrivește cu mai multe reguli generate de fișiere malware, atunci este considerat posibil malware. În cadrul modulului de învățare automată, atributele extrase ale fișierului analizat sunt furnizate modelului antrenat pentru a obține un rezultat. La final, după ce ambele module au generat răspunsuri în urma analizei fișierului, aceste răspunsuri sunt furnizate utilizatorului.

Propunem o arhitectură modulară, în care datele să fie vehiculate prin intermediul serviciului de e-mail Gmail, urmând a fi analizate automat în mediul Cuckoo. Rezultatele obținute prin această analiză vor fi stocate în baza de date a unei platforme de partajare a informațiilor de securitate – MISP, cu scopul de a asigura un nivel de control mai înalt asupra fișierelor utilizate pentru testare. Prin intermediul acestei abordări, se poate realiza o gestionare mai eficientă a procesului de testare și se poate optimiza calitatea și securitatea soluției dezvoltate.

Capitolul 2. Proiectarea aplicației

2.1. Arhitectura aplicației

Lucrarea propune elaborarea unei aplicații care să efectueze analiza unui fișier prin examinarea comportamentului său cu scopul de a determina dacă acesta reprezintă sau nu o amenințare.

Aplicația este concepută cu o structură modulară (figura 2.1), permițând adăugarea facilă a unor module noi în procesul de dezvoltare. Abordarea modulară oferă flexibilitate în ceea ce privește funcționalitatea aplicației. Fiecare modul îndeplinește o anumită funcție sau sarcină specifică, precum analiza de comportament, extragere de caracteristici, generare de reguli sau antrenare de modele.

Fluxul aplicației pornește de la modulul de extragere a atașamentelor de e-mail care sunt extrase și ulterior trimise către Cuckoo *sandbox*. După finalizarea analizei, se generează un raport detaliat, care este trimis atât către platforma MISP, cât și către modulul de extragere de caracteristici. Apoi aplicația parcurge două etape distincte pentru analiza și clasificarea datelor. În prima etapă, sunt generate reguli Yara pe baza datelor primite de la modulul de extragere de caracteristici și modelele de învățare automată sunt antrenate . Această etapă are scopul de a dezvolta modele și reguli eficiente care să poată identifica și clasifica corect diversele tipuri de date analizate. Este important de menționat că procesul de extragere a caracteristicilor diferă pentru cele două module. Urmează etapa a doua în care aplicația intră în funcțiune, unde pe baza modelelor antrenate și regulilor generate, se ia o decizie în ceea ce privește caracterul potențial malițios al datelor analizate. Rezultatele obținute de cele două module sunt centralizate și trimise către utilizatorul serviciului de e-mail pentru informare.

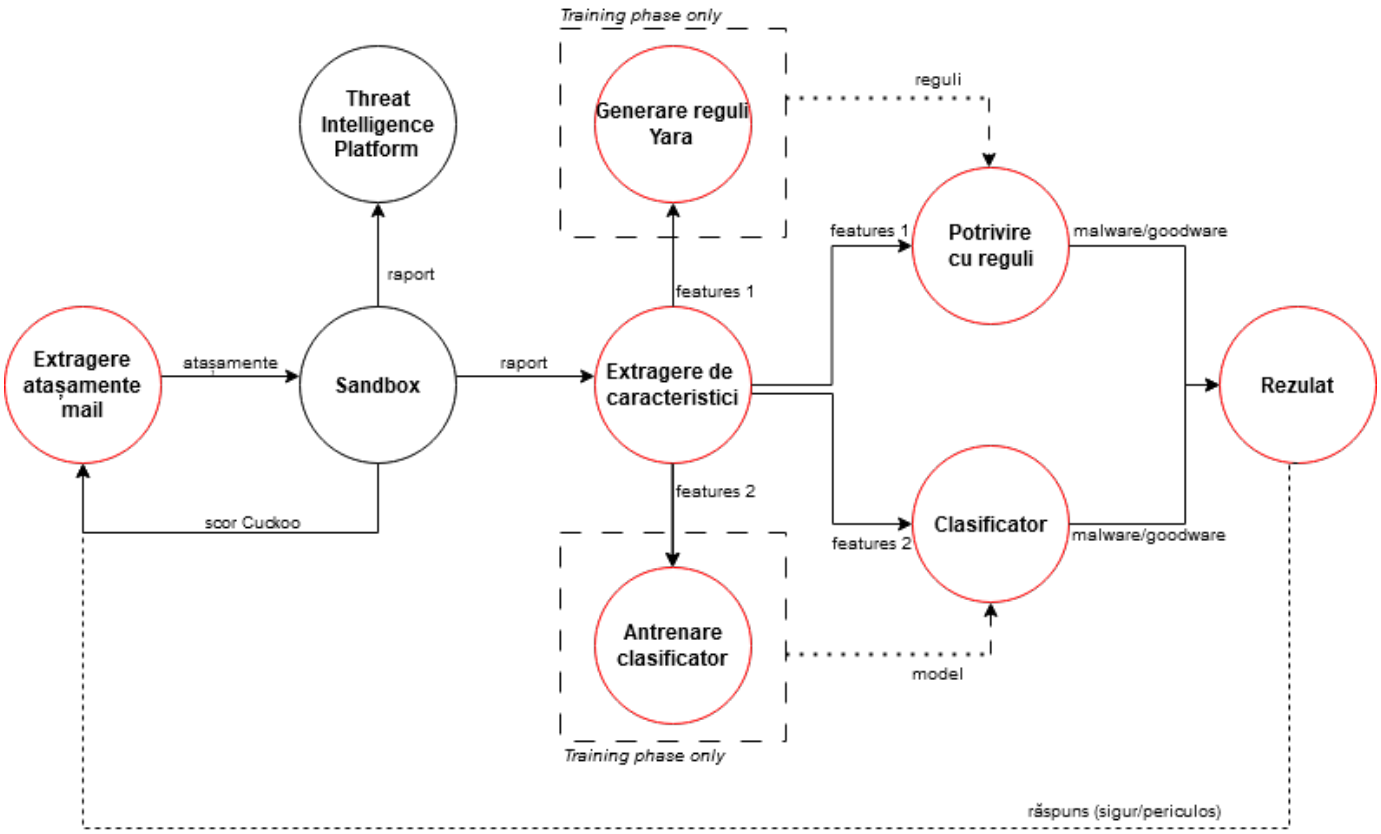

Figura 2.1. Flux de funcționare a aplicației.

Pentru a permite comunicarea între modulele respective, s-a utilizarea *framework*-ului Flask. Alegerea acestui *framework* a fost determinată de necesitatea de a personaliza *sandbox*-ul și de a facilita integrarea de noi funcționalități în mod simplu. Modulele implementate returnează răspunsuri sub formă de dicționare, acest format este convenabil deoarece Flask oferă suport nativ pentru serializarea și deserializarea datelor în format JSON. Astfel, transmiterea și recepționarea datelor în format JSON între module devine ușoară și eficientă.

2.2. Modulul de extragere a atașamentelor din e-mail

Serviciul de e-mail utilizat în cadrul acestei lucrări este Gmail, aceasta integrare se face doar pentru a demonstra funcționalitatea aplicației într-un caz real (de exemplu integrarea cu un serviciu de e-mail al unei instituții). Ținând cont de faptul ca în Gmail nu se pot trimite fișiere executabile deja riscul pentru infectare de malware se minimizează, astfel modulul este proiectat ca atunci când clientul primește un e-mail sa recepționeze ca răspuns doar scorul oferit de Cuckoo după analiza sa. Deși proiectarea aplicației permite analizarea atașamentelor prin intermediul modulelor proprii este important de menționat că aceste module funcționează exclusiv pentru fișiere executabile (programe Windows și *ransomware*). Astfel, pentru alte tipuri de fișiere sau formate, modulele nu sunt aplicabile și prin urmare nu au sens în contextul analizei și evaluării acestora.

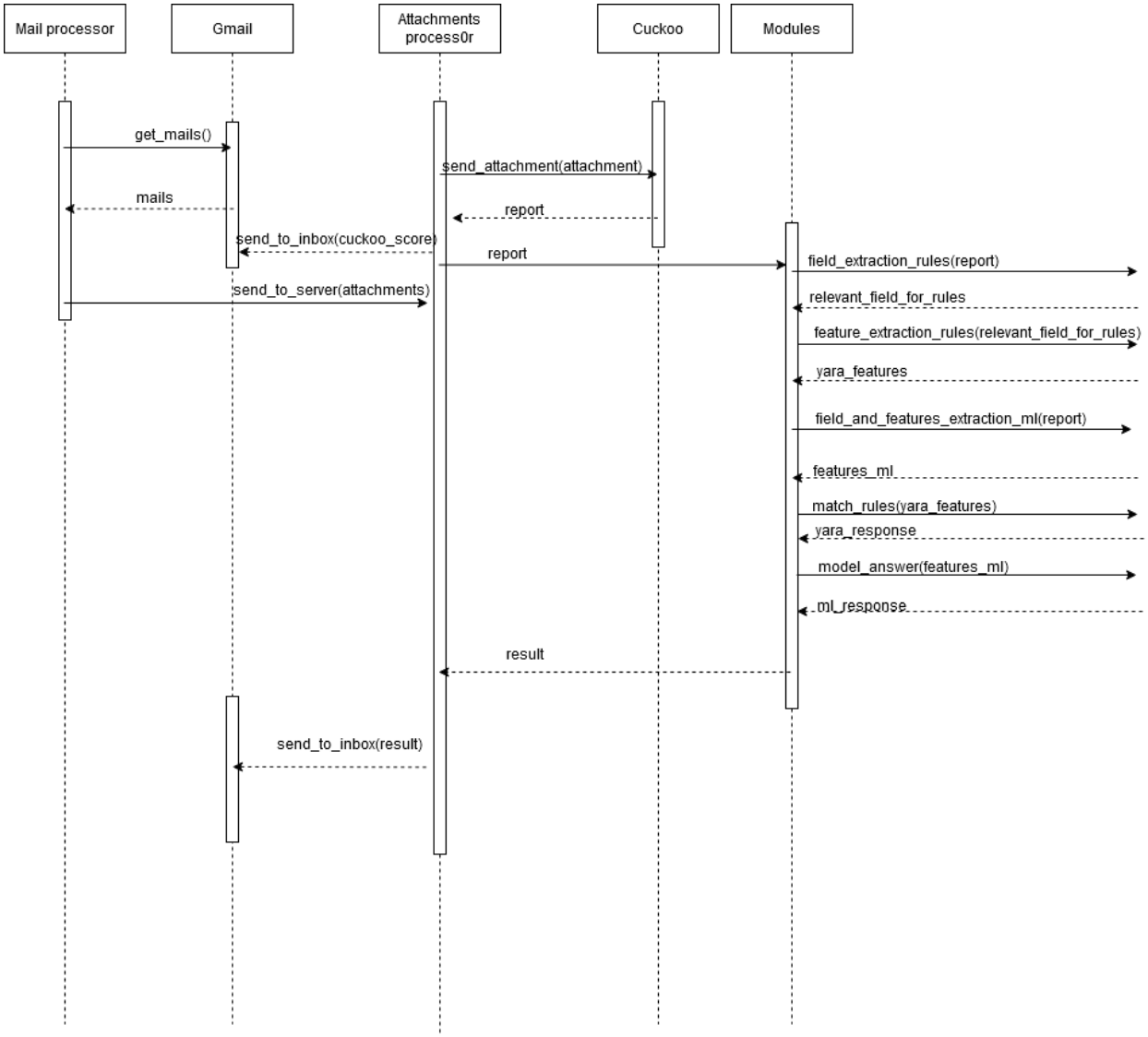

Figura 2.2. Diagrama de secvență pentru funcționarea aplicației.

În Figura 2.2 este prezentată interacțiunea soluției începând de la modulul de extragere a

atașamentelor de e-mail, integrat prin intermediul Gmail API. Acest modul parcurge odată toate e-mail-urile din inbox-ul unui client și extrage toate atașamentele existente, urmând ca ulterior să analizeze doar e-mail-urile noi. Acestea sunt ulterior trimise către *sandbox* pentru a fi analizate și pentru a obține un scor determinat de *sandbox*. După finalizarea analizei, raportul este trimis către modulele de extragere a caracteristicelor, care procesează raportul și returnează trăsăturile corespunzătoare. Aceste caracteristici sunt apoi trimise către modulul de potrivire cu reguli Yara și modulul de clasificare. Modulele respective furnizează fiecare câte un rezultat iar acestea sunt procesate astfel încât, în cazul în care fișierul este considerat sigur conform modulului Yara, răspunsul clasificatorului să fie influențat de această decizie. Astfel, se furnizează trei răspunsuri către client: decizia modulului Yara, decizia clasificatorului și decizia clasificatorului influențată de răspunsul oferit de modulul Yara.

Prin intermediul acestui mecanism, utilizatorul obține informații relevante despre natura și potențialul malițios al atașamentelor primite prin intermediul serviciului de e-mail.

2.3. Modulul Cuckoo

Sandbox-ul Cuckoo este integrat cu modul de extragere de atașamente, MISP și cel de extragere de caracteristici, pentru a realiza acest lucru a fost utilizat API-ul de care dispune Cuckoo pentru a furniza fișiere noi pentru analiza și a prelua rapoartele generate.

Sandbox-ul Cuckoo este integrat cu modulul de extragere a atașamentelor, MISP și extragere de caracteristici, realizând astfel o colaborare între aceste module prin intermediul API-ului furnizat de Cuckoo[10]. Prin intermediul acestui API, modulele pot interacționa și partaja informații esențiale pentru analiza și detectarea amenințărilor. Modulul de extragere a atașamentelor utilizează API-ul pentru a trimite fișierele noi către *sandbox*-ul Cuckoo, iar rapoartele generate în format JSON sunt accesate și utilizate de celelalte module prin intermediul API-ului. Această integrare prin API facilitează schimbul de informații și funcționalități între module, asigurând o colaborare eficientă și sincronizată în cadrul aplicației. Deși Cuckoo dispune de un API dezvoltat lipsește funcționalitatea de notificare a utilizatorului când un fișier a fost terminat de analizat, astfel aceasta nevoie este implementată. Integrarea *sandbox*-ului cu platforma MISP este realizată cu ușurință datorită suportului oferit de *sandbox* pentru această integrare. Configurarea necesară pentru realizarea acestei integrări este simplă.

Raportul generat de Cuckoo este unul complex, acesta este structurat în general în secțiunile prezentate în Figura 1.5. Aceste secțiuni sunt esențiale pentru a furniza informații detaliate despre analiza și rezultatele obținute. Rolul fiecărei secțiuni fiind:

- Info: informații generale despre desfășurarea analizei
- Procmemory: informații despre secțiuni ale memoriei dintr-un proces sau dintr-un fișier
- Target: informații despre fișierul analizat (nume, sha256, md5, marime)
- Network: informații despre traficul de internet (protocoale folosite, adrese ip, hosts)
- Signatures: modele suspecte, conține si indicatori de compromitere
- Behavior: informații despre procese și API-uri
- Debug: reprezintă informații suplimentare despre procesul de analiză
- Metadata: conține informații despre fișiere generate de analiză (ex.: pcap)

Pentru arhitectura propusă și analiza fișierelor pe baza apelurilor api și a traficului de internet secțiunile de interes sunt reprezentate de `Network` și de `Behavior` care au la rândul lor subsecțiuni și atribute (ex. `apistats`), detaliate în Figura 2.3.

[10]Mai multe detalii despre Cuckoo API: https://cuckoo.readthedocs.io/en/latest/usage/api/

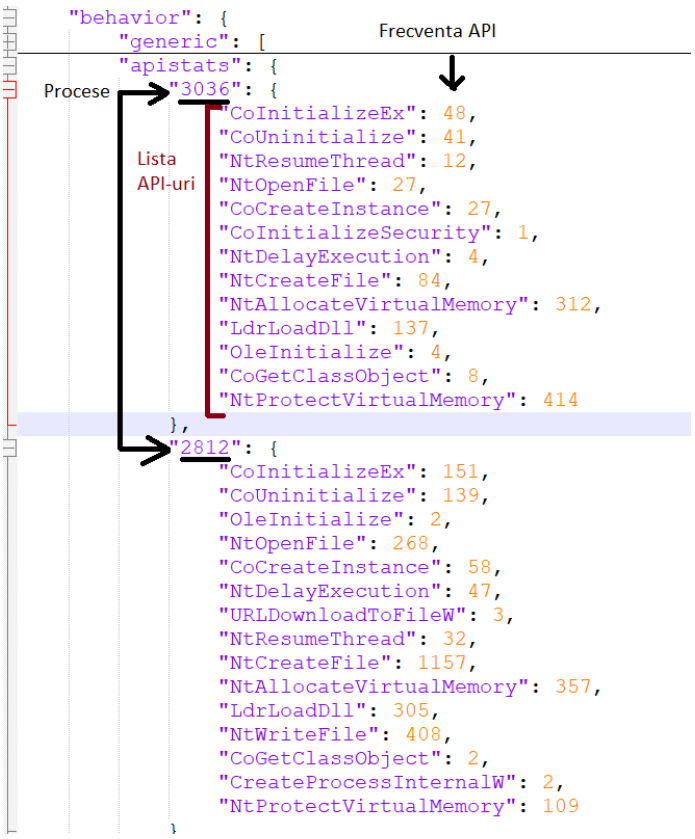

Figura 2.3. Cuckoo raport listă API-uri

Secțiunea `Network` furnizează informații despre comunicarea rețelei efectuată de fișierul analizat. Aici pot fi găsite adrese IP, porturi și protocoale utilizate în timpul analizei. Aceste informații pot dezvălui activități suspecte sau conexiuni cu servere malware. Secțiunea `Behavior` prezintă detaliile despre comportamentul fișierului în timpul analizei. Aceasta include apelurile API efectuate, procesele create și resursele utilizate. Prin analizarea acestor informații, se pot identifica acțiuni suspecte sau comportamente anormale care pot indica prezența unui malware.

2.4. Modulul MISP

Importanța corelării datelor într-o platformă de tip Threat Intelligence a fost descrisă în capitolul anterior, în subcapitolul 1.2.5.

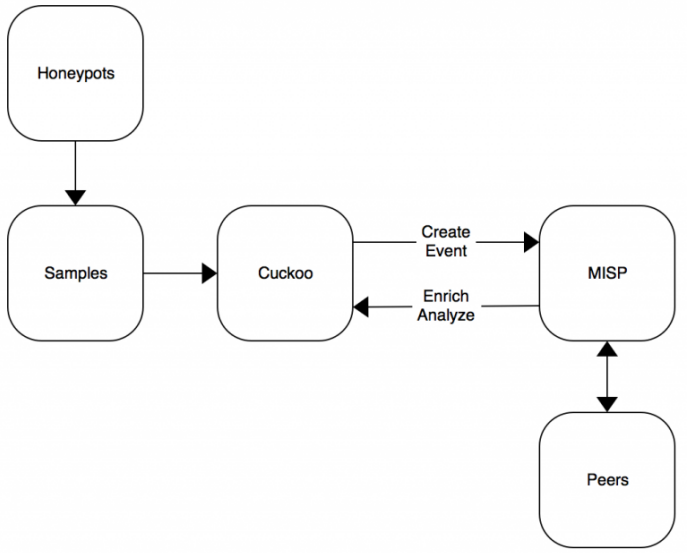

Figura 2.4. Integrarea *sandbox* Cuckoo cu platforma MISP, conform documentației[11] MISP

În Figura 2.4 se observă că integrarea *sandbox*-ului Cuckoo cu platforma MISP este posibilă în două moduri:

- Rezultatele analizei Cuckoo sunt îmbogățite cu IOC-urile găsite în MISP
- IOC-urile găsite în eșantion sunt corelate cu MISP și sunt afișate ID-ul evenimentului, descrierea și nivelul alertei

În cadrul acestei lucrări de diplomă, funcționalitatea este implementată prin integrarea *sandbox*-ului Cuckoo cu platforma MISP pe direcția în care rezultatele analizei Cuckoo sunt îmbogățite cu IOC-urile găsite în MISP. După fiecare fișier analizat, raportul rezultat este trimis către motorul de corelare MISP pentru a stoca datele relevante în cadrul platformei.

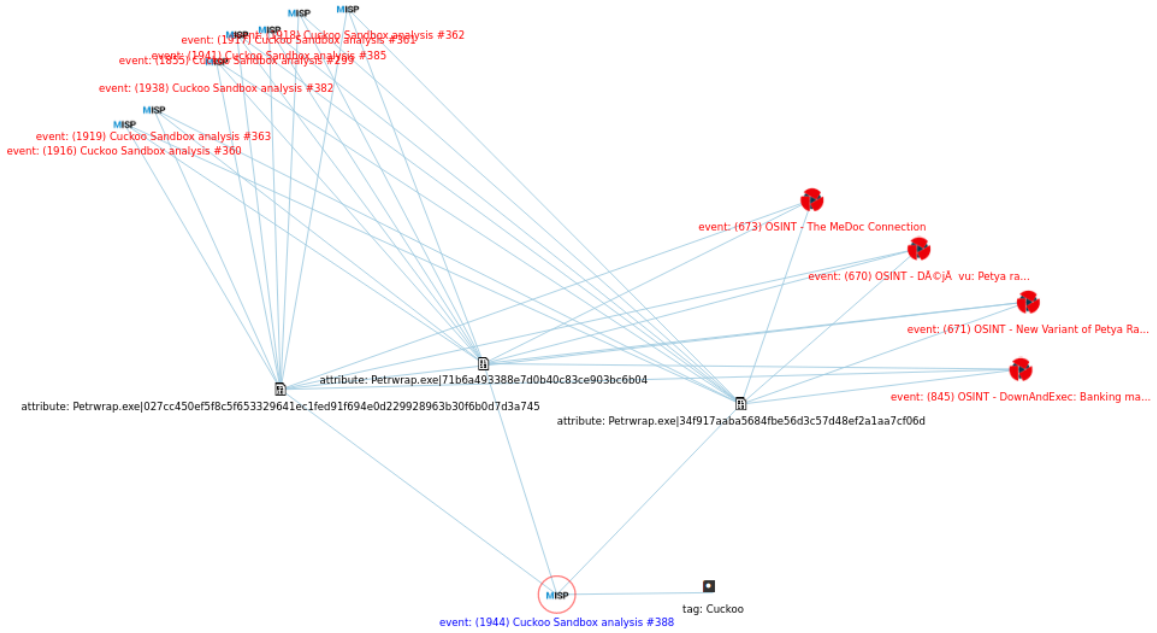

Figura 2.5. Exemplu de corelare a unui fișier analizat de Cuckoo în MISP.

În Figura 2.5 se ilustrează procesul de corelare a unui fișier analizat de Cuckoo cu alte evenimente. Fișierul analizat este reprezentat prin scrisul albastru și se află în partea de centru-jos al imaginii. Acesta este corelat cu alte fișiere analizate de Cuckoo (reprezentate în partea de sus-stânga a imaginii) și cu evenimentele raportate de alte organizații (reprezentate în partea dreaptă a imaginii). Aceste corelări se realizează prin identificarea a trei atribute comune. Reprezentarea grafică facilitează înțelegerea comportamentului fișierului analizat, evidențiind legăturile acestuia cu alte evenimente și fișiere analizate.

2.5. Modulul de extragere de caracteristici

În cadrul acestei soluții, sunt utilizate două metode distincte pentru determinarea intenției unui fișier, ceea ce conduce la utilizarea a două modalități de extragere a caracteristicelor. Această abordare permite obținerea de informații relevante și diverse despre fișierul analizat, utilizând tehnici și algoritmi specifice fiecărei metode. Prin combinarea rezultatelor obținute din cele două modalități de extragere a caracteristicelor, se obține o imagine mai completă asupra naturii și comportamentului fișierului în cauză.

[11] https://blog.rootshell.be/2017/01/25/quick-integration-misp-cuckoo/

2.5.1. Extragere de caracteristici pentru reguli Yara

Pentru a antrena și testa modulul de detectare a malware-urilor pe baza regulilor Yara este necesară extragerea caracteristicilor semnificative din raportul generat de *sandbox*. În vederea obținerii acestui raport, fișierul suspect este introdus în *sandbox*-ul Cuckoo, unde va fi executat într-un mediu izolat și controlat pentru a obține date cu privire la comportamentul său.

În vederea selectării caracteristicilor relevante, se filtrează datele extrase din secțiunile `behavior` și `network` pentru a identifica doar aspectele care sunt semnificative pentru analiza comportamentală a fișierelor potențial malițioase. În această etapă obiectivul este de a identifica adresele IP și DNS-urile utilizate, precum și de a extrage informații despre toate API-urile folosite de fiecare proces și de a identifica API-urile în ordinea apelării de fiecare proces ale malware-ului. După prelucrarea raportului generat de *sandbox*, sunt utilizați doi algoritmi pentru a determina importanța atributelor: TF-IDF și LCS.

Aceasta abordare de utilizare a algoritmului TF-IDF pentru extragerea celor mai relevante caracteristici ale unui fișier, precum apelurile API și adresele IP și DNS-uri din traficul de internet, este foarte utilă în identificarea elementelor cu impact și eliminarea apelurilor API care pot afecta analiza (zgomot). Prin aplicarea algoritmului TF-IDF, se poate acorda o pondere mai mare apelurilor API și adreselor IP și DNS-uri care sunt mai specifice pentru fișierul analizat, în timp ce se reduc ponderile pentru cele mai comune și mai generice. Astfel, se obține o selecție mai precisă a caracteristicilor relevante și se minimizează influența zgomotului sau a elementelor nesemnificative. Aceasta permite o analiză mai eficientă a fișierelor și o identificare mai precisă a comportamentului malițios.

Algoritmul TF-IDF este aplicat pentru identificarea apelurilor API cu o relevanță mai mare, în timp ce algoritmul LCS este folosit pentru a identifica comportamentul predominant al fișierului prin găsirea de secvențe comune de apeluri API între procese. Acest aspect este crucial deoarece, în opoziție cu un apel API izolat, o secvență de apeluri API oferă o perspectivă mai clară asupra intenției acestora.

Primul algoritm utilizat este TF-IDF (Term Frequency - Inverse Document Frequency), care are ca scop determinarea relevanței cuvintelor cheie dintr-un document. În cazul de față, documentul este reprezentat de lista de API-uri și frecvențele acestora pentru fiecare proces, precum și adresele IP și DNS-urile. Algoritmul TF-IDF are doua etape: `Term Frequency` și `Inverse Document Frequency`.

În prima etapă, `Term Frequency` (`TF`) se calculează conform Ecuației 2.1, indicând astfel importanța apelurilor de sistem pentru fiecare proces.

$$\text{TF} = \frac{\text{nr. de apariții a unui API pentru un proces}}{\text{nr. total de API-uri pentru un proces}} \tag{2.1}$$

La fel este aplicat și peste adresele IP și DNS-uri. Pentru a determina importanța API-urilor pentru un fișier potențial malițios, este necesar să se determine importanța apelurilor pentru toate procesele, nu doar pentru unul.

`Inverse Document Frequency` (`IDF`) reprezintă cea de a doua etapă a algoritmului, în care se determină importanța unui API în funcție de toate API-urile apelate de toate procesele. `IDF` se calculează conform Ecuației 2.2:

$$\text{IDF} = \log\left(\frac{\text{Număr total de procese}}{\text{Număr de procese în care apare API-ul}}\right) \tag{2.2}$$

Înmulțind `TF` cu `IDF`, se obține relevanța finală a unui API (Ecuația 2.3). Cu cât valoarea este mai mare, cu atât este mai important astfel, in cadrul acestei lucrări, pentru identificarea apelurilor API, adreselor IP și DNS-urilor relevante, se selectează primele trei caracteristici cu scorul cel mai mare în ceea ce privește apelurile API și traficul de internet

$$\text{TF-IDF} = \text{TF} \times \text{IDF} \tag{2.3}$$

Al doilea algoritm utilizat este LCS (Longest Common Subsequence) și este aplicat peste secvența de API-uri în ordinea apelării pentru fiecare proces. LCS are ca scop determinarea secvenței comune de apeluri API dintre două procese. Fiind mai multe procese, acestea sunt luate prin combinări, iar la final se va alege cea mai lungă secvență comună. De exemplu, dacă un malware are două procese, primul proces cu API-urile: X, Y, Z, M iar al doilea proces cu API-urile: Y, H, H, Z, O, M, rezultatul va fi YZM.


```
"behavior": {
  "generic": [],
  "apistats": {
    "308": {
    },
    "604": {
    }
  },
  "processes": [
  {
    "pid": 308,
    "process_path": "C:\\Temp\\df5417b160e36b...",
    "calls": [
    {
      "category": "system",
      "status": 1,
      "stacktrace": [],
      "api": "LdrGetDllHandle",      <-----------
      "return_value": 0,
         "arguments": {
            "module_name": "KERNEL32.DLL",
            "module_address": "0x7c800000"
         },
      "time": 1496523552.552217,
      "tid": 396,
      "flags": {}
    } ......
   ] ......
  },
  {
    "pid": 604,
    "process_path": "\\??\\C:\\WINDOWS\\system32\\csrss.exe",
    "calls": [
    { ......
    }
    ......
  }
  ]
}
```


Listing 2.1. Cuckoo raport pentru algoritmul LCS

În Listing 2.1 sunt afișate o parte din subsecțiunile raportului generat de Cuckoo pentru aplicarea algoritmului LCS asupra unui ransomware Hydracrypt[12], a cărui hash SHA-256 este:

`df5417b160e36b7c7417277e46aefa82`
`f212fe7bc08f0f508c03e0108375d0ca`

Din secțiunea `apistats` se poate observa că acest ransomware conține două procese (cu `pid`-urile 308 și 604), iar acestea sunt descrise în secțiunea `processes`. Totuși, secțiunea de interes pentru aplicarea algoritmului LCS este secțiunea `calls` pentru fiecare proces. Aplicarea algoritmului

[12] https://www.virustotal.com/gui/file/df5417b160e36b7c7417277e46aefa82f212fe7bc08f0f508c03e0108375d0ca/detection

LCS pe mai multe procese conduce la obținerea unei secvențe lungi de API-uri. În Listing 2.2 este prezentat un exemplu de caracteristici extrase pentru ransomware, iar în secțiunea "LCS" este stocat rezultatul aplicării algoritmului pe fișierul respectiv. Din Listing 2.1 se poate observa că primul apel realizat de procesul 308 este `LdrGetDllHandle` (din subsecțiunea `api`) iar în Listing 2.2 se observa ca primul apel comun al celor două procese este același API.

```
{
  "API": [
    ["SizeofResource", 0.12985607656093306],
    ["GetKeyState", 0.1074766184307713],
    ["LoadResource", 0.06492803828046653]
  ],
  "LCS": [
    "LdrGetDllHandle",          <-----------
    "LdrGetProcedureAddress",
    "LdrGetProcedureAddress",
    "LdrGetProcedureAddress",
    "LdrGetDllHandle",
    "LdrGetProcedureAddress",
    ".....",
    "LdrGetProcedureAddress",
    "NtAllocateVirtualMemory",
    "LdrGetDllHandle",
    "NtFreeVirtualMemory",
    "LdrGetDllHandle",
    "LdrGetDllHandle",
    "NtClose"
  ],
  "NETWORK": [
    "2.17.148.103",
    "104.105.48.82",
    "191.239.213.197",
    "2.edgekey.net",
    "2.edgekey.net.globalredir.akadns.net",
    "e1863.dspb.akamaiedge.net"
  ]
}
```

Listing 2.2. Caracteristici pentru *ransomware hydracrypt* cu două procese.

După ce au fost extrase caracteristicile, în faza de antrenare, acestea sunt stocate într-o bază de date Redis pentru a putea crea apoi reguli Yara pe baza lor, urmând ca apoi, în faza de testare doar să fie extrase pentru a se putea efectua verificări pe regulile respective. Baza de date Redis este utilizată exclusiv în faza de antrenare și testare a modulului. Atunci când aplicația rulează în mod obișnuit, în timpul funcționării obișnuite a aplicației, datele nu sunt stocate în nicio bază de date permanentă. În schimb, ele sunt pur și simplu transmise și gestionate prin intermediul *framework*-ului Flask.

2.5.2. Extragere de caracteristici pentru model

În procesul de antrenare al modulului de învățare automată, sunt selectate caracteristici diferite față de cele utilizate pentru generarea regulilor, cu scopul de a obține o imagine mai completă asupra analizei fișierului. Aceste caracteristici sunt extrase din raportul furnizat de Cuckoo. În cazul generării regulilor Yara, trăsăturile relevante includ traficul de internet, apelurile API și secvențele comune de API între două procese. În schimb, pentru a analiza problema dintr-o altă perspectivă, se utilizează doar procesele care sunt direct derivate din procesul părinte.

Prin aceasta abordare se analizează subprocesele care au legătură directă cu procesul părinte deoarece acestea pot fi mai relevante în anumite cazuri. Atunci când un proces creează

subprocese, există o relație ierarhică între acestea, iar comportamentul subproceselor poate fi influențat de procesul părinte. Analizând subprocesele, se poate înțelege propagarea comportamentului și posibilele activități maligne în cadrul ierarhiei de procese. Totodată se pot obține informații despre acțiunile specifice și interacțiunile care au loc în cadrul contextului procesului părinte. Aceste informații pot ajuta la identificarea de tipare, detectarea de anomalii și descoperirea de amenințări potențiale care pot fi greu de observat prin analiza proceselor individuale izolat. Pentru a realiza acest lucru, se examinează raportul generat de Cuckoo și se identifică procesele care sunt copii direcți ai procesului părinte, acest lucru este realizat prin analizarea secțiunii din `behavior\processtree`. Ulterior, se extrag secvențele de API-uri relevante pentru fiecare proces în parte din `behavior\processes\calls`. Odată ce sunt preluate procesele și secventele API urmează o procesare pentru a prelucra trăsăturile pentru modelele de învățare automata.

Astfel, în cazul în care un fișier are un proces copil direct asociat procesului părinte, se urmărește obținerea unei secvențe mai scurtă a apelurilor API (dacă există) prin selectarea doar a primelor 100 de apeluri care nu sunt consecutive. În Listingul 2.3 este prezentat un exemplu de procesare a unei secvențe de API-uri pentru un proces, conform acestei abordări.

```
Secventa intitiala: .., NtClose, NtClose, NtMapViewOfSelection, NtClose,
 ↪  ....
Secventa dupa procesare: .., NtClose, NtMapViewOfSelection, NtClose, ....
```

Listing 2.3. Aplicare procesare pe caracteristici

După extragerea acestor secvențe pentru fiecare proces copil, ele sunt codificate într-un format numeric. Această codificare numerică se aplică deoarece setul de date folosit pentru antrenarea modelului conține 307 apeluri API reprezentate numeric. Astfel, aceeași metodă de codificare este aplicată atât pe datele de antrenare, cât și pe intrările noi.

Este important să menționăm că cele două modalități de extragere a caracteristicilor au utilizat codificări diferite datorită antrenării lor pe seturi de date distincte. În primul rând, pentru generarea regulilor Yara, o codificare numerică nu ar fi fost potrivită, deoarece reprezentarea frecvenței în astfel de format, după ce apelurile API sunt codificate numeric, poate fi confuză și dificil de interpretat. În al doilea rând, maparea inversă a caracteristicilor pentru modulul de învățare automată ar fi necesitat un pas suplimentar care nu este necesar în acest caz.

Diferența abordării prezentate în această lucrare față de alte abordări existente constă în faptul că un fișier nu rezultă într-o singură secvență API, ci, în cazul în care există mai multe procese copii care provin direct din procesul părinte, fiecare secvență este procesată în asociere cu fișierul analizat și este trimisă către partea de antrenare/utilizare a sistemului.

2.6. Modulul de antrenare a modelelor

Faza de antrenare constă în preluarea trăsăturilor din modulul anterior și transmiterea acestora către mai multe modele. Algoritmii de învățare automată au fost utilizați în detectarea malware-urilor. Algoritmi precum K-Nearest Neighbor (KNN), Decision Tree (DT), Naive Bayes (NB) și Support Vector Machine (SVM) sunt considerați cei mai des utilizați algoritmi în detectarea malware-urilor [25]. Din acest motiv pentru alegerea unui model potrivit am decis testarea algoritmilor KNN, SVM, Random Forest (RF) și Logistic Regression (LR). La final fiind ales modelul cu acuratețea cea mai mare.

K-nearest neighbors este o metodă de învățare automată supervizată utilizată în principal pentru clasificare, dar poate fi folosită și pentru regresie. Acest algoritm se bazează pe ideea că obiectele similare sunt în general mai apropiate în spațiul caracteristicilor. Există două variante ale algoritmului KNN: cea cu distanță simplă și cea cu distanță ponderată. Diferența dintre cele două constă în modul în care se efectuează clasificarea. Varianta cu distanță simplă se bazează exclusiv pe distanța între punctele din spațiul caracteristicilor pentru a determina clasa unei noi instanțe.

În schimb, varianta cu distanță ponderată ia în considerare și cat de apropiat este fiecare vecin în procesul de clasificare, acordând o pondere mai mare vecinilor mai apropiați. Având în vedere ca setul de date conține valori intre 1 și 308 datele au fost normalizate pentru a avea o reprezentare corecta. Numărul de vecini a fost determinat experimental, iar pentru calcularea distanței s-au evaluat trei opțiuni: distanța Euclidiană, distanța Manhattan (potrivită pentru modele cu mai multe caracteristici) și similaritatea cosinus (folosită pentru documente sau secvențe de gene). Din cele trei opțiuni, a fost selectată metoda care a obținut cea mai mare acuratețe.

Support Vector Machine este o alta metoda de învățare automată supervizată utilizată în principal pentru clasificare. Acest algoritm funcționează prin găsirea celui mai bun hiperplan pentru a separa datele și a le clasifica, iar regularizarea automată a datelor face ca algoritmul să fie potrivit și pentru probleme cu dimensionalitate mare. Pentru găsirea configurației optime în cadrul modelului Support Vector Machine (SVM), s-au ajustat parametrii următori în biblioteca `scikit-learn`: C, funcția de kernel, gamma și gradul (în cazul kernelului polinomial). Configurația optimă rezultată a fost determinată empiric prin explorarea diferitelor combinații de parametri.

Logistic Regression este un algoritm de învățare automată supervizată utilizat în principal pentru clasificare. Acest algoritm se concentrează pe evaluarea probabilității de apartenență a unei instanțe la o clasă specifică, folosind funcția sigmoidă. Este preferat datorită simplității sale și a capacității de a fi folosit în clasificarea binară, fiind adesea utilizat în lucrări care implică detectarea de malware. Abordarea pentru testarea acestui algoritm este similară cu cea menționată anterior, iar intrările sunt scalate pentru a asigura consistența între caracteristici. Pentru găsirea configurației optime în cadrul acestui model, s-au ajustat parametrii precum C, penalty și solver din biblioteca `scikit-learn`.

Random Forest, menționat în 1.2.6, este un algoritm de învățare automată supervizată care se evidențiază în această lucrare prin precizia sa, așa cum este evidențiat în rezultatele prezentate în tabelul 2.1. Pentru alegerea modelului optim, s-a utilizat metoda de validare încrucișată stratificată (*stratified cross-validation*), unde algoritmul Random Forest a obținut cele mai bune rezultate. Având în vedere că modelul de analiză implică un set de date cu 100 de caracteristici, Random Forest s-a dovedit a fi eficient în gestionarea unei astfel de dimensionalități ridicate. Pentru a determina parametrii optimi ai algoritmului, s-a realizat o analiză empirică, în care s-au ajustat și testat diferite combinații de parametri pentru a obține cea mai bună performanță.

Tabelul 2.1. Rezultate obținute pentru modulul de învățare automată

	RF	**SVM**	**KNN**	**LR**
Accuracy	0.897476	0.815457	0.862776	0.809148
F1-score	0.8987	0.8152	0.8612	0.8220
Precision	0.9444	0.7963	0.8333	0.8056

În concluzie, fiecare dintre modelele menționate anterior (KNN, SVM, Logistic Regression și Random Forest) are avantaje și dezavantaje distincte. Prin testarea și evaluarea lor în cadrul detecției de malware, s-a căutat alegerea potrivită pentru obținerea rezultatelor optime.

2.7. Modulul de generare a regulilor Yara

Regulile YARA vor fi generate doar în faza de antrenare. Pentru un potențial malware se vor crea trei reguli, fiecare regulă corespunzând unei anumite trăsături, anume: adrese IP și DNS-uri, apeluri de API și secvente de API. Primele două caracteristici sunt preluate din baza de date Redis și sunt cele procesate prin algoritmul TF-IDF, ultima caracteristică fiind cea prelucrată prin LCS. După generarea regulilor, acestea sunt împărțite în două categorii în funcție de verdictul generat de Cuckoo: periculoase și inofensive. Raportul generat de *sandbox* conține un scor între

0 și 10, care indică nivelul de amenințare a malware-ului, zero indicând că este inofensiv iar zece indicând că este extrem de periculos (Listing 2.4 unde *type* reprezintă scorul).

```
rule TF_IDF_API_filename
{
    meta:
        type = "4.6"
    strings:
        $ = "GetSystemMetrics"
        $ = "NtEnumerateValueKey"
        $ = "GetKeyState"
    condition:
        any of them
}
```

Listing 2.4. Regulă Yara generată din TF-IDF pentru API cu scor 4.6

În procesul de generare a regulilor Yara din trăsăturile extrase prin algoritmul Longest Common Subsequence (LCS), s-a constatat o problemă. Pentru unele mostre de malware, lungimea regulilor obținute era foarte mare, ceea ce poate avea un impact negativ asupra performanței sistemului de detecție. Pentru a depăși această problemă, s-a folosit o tehnică de codificare a apelurilor de sistem utilizate în reguli. Astfel, în loc de a include toate apelurile de sistem, acestea au fost codificate iar dacă o anumită secvență de apeluri era consecutivă se adăuga și frecvența acestora în regulă. De exemplu, o regulă care inițial ar fi arătat ca `NtOpenProcess NtOpenProcess NtClose` arată în felul următor după aplicarea codificării: `NOP 1 NC`. Această abordare a permis reducerea lungimii regulilor (Listing 2.5) și implicit, creșterea performanței sistemului de detecție.

```
rule LCS_filename
{
    meta:
        type = "4.6"
    strings:
        $ = "GSM KC NEVK GKS LGPA 1 NWF 19"
    condition:
        any of them
}
```

Listing 2.5. Exemplu regulă Yara codificată

Regulile Yara sunt create doar în timpul fazei de antrenare iar apoi sunt aplicate în faza de testare. Acestea reprezinta o modalitate simplă de verificare a unui potențial malware, se pot genera mereu mai multe reguli însa necesită mai multe date de antrenament. Aceste reguli au fost create pe un set de date foarte divers pentru a putea observa cum se comportă într-un caz real. Problema la aceasta abordare este ca fiecare familie de malware-uri are alt tip de comportament, de exemplu un keylogger urmărește ce tastează victimele și utilizează apeluri de sistem specifice tastaturii, cum ar fi `GetAsyncKeyState` și `SetWindowsHookEx`. Pe când un worm urmărește să se replice singur și are operații de citire și scriere de procese, cu apeluri de sistem precum `ReadProcessMemory`, `WriteProcessMemory`. Astfel, dacă setul de date inițial este foarte aleatoriu sau mic, este posibil sa se genereze reguli care nu se potrivesc.

Astfel pentru a testa soluția, setul de date de antrenare consta doar în fișiere executabile Windows pentru clasa de inofensiv[13] și pentru clasa doar malware-uri care aparțin familiei Ransomware[14].

[13] https://drive.google.com/drive/folders/1juNrSMY8lQHzfwI7YVlW1yiJId8j1H-9
[14] https://goo.gl/e8jbXq

2.8. Modulul de potrivire cu reguli Yara

Pentru a ajunge la un verdict acest modul are rolul de a compara caracteristicile extrase (conform 2.5.1) cu cele existente din Yara. Modalitatea prin care se mărește acuratețea este ca decizia sa fie făcută pe bază de vot majoritar. Astfel, există trei cazuri:

1. În cazul în care caracteristicile se potrivesc cu mai multe reguli din categoria de malware, verdictul este că acesta este malware.

2. În situația în care caracteristicile se potrivesc cu mai multe reguli din categoria de fișiere inofensive, verdictul este că acesta nu este rău intenționat.

3. Dacă caracteristicile se potrivesc cu un număr egal de reguli din ambele categorii, se consideră că acesta este malware.

În situația în caracteristicile se potrivesc cu un număr egal de reguli din ambele categorii, s-a decis că este mai bine să fie clasificat ca malware, în loc să existe riscul de a infecta sistemul.

2.9. Modulul de clasificare

După finalizarea etapei de antrenare, datele extrase din raport sunt prelucrate conform metodei descrise în 2.5.2 și apoi furnizate către modelul antrenat. Fiecare secvență API care aparține fișierului este evaluată individual, astfel dacă un fișier conține doua secvențe API care au rămas după procesarea datelor, modelul le va analiza pe fiecare în mod individual. Rezultatul modulului este în forma vectorială, valoarea 1 reprezentând categoria de malware iar valoarea 0 – goodware (fișier inofensiv) , astfel, pentru un fișier care are doar o secvență API după procesarea datelor, rezultatul va fi 1 sau 0. Însă, în cazul în care fișierul conține două secvențe API, rezultatul poate fi reprezentat sub forma unui vector, cum ar fi [0,0] (ambele secvențe sunt inofensive), [1,0] (prima secvență este suspectă), [0,1] (a doua secvență este suspectă) sau [1,1] (ambele secvențe sunt suspecte). Această reprezentare vectorială permite identificarea precisă a categoriei corespunzătoare fiecărei secvențe API în cadrul fișierului analizat.

Capitolul 3. Implementarea aplicației

3.1. Modulul de extragere a atașamentelor din e-mail

Pentru a prelua atașamentele din Gmail în cadrul acestei lucrări, am utilizat Gmail API prin intermediul unor biblioteci furnizate de Google, cum ar fi `google.auth.transport.requests` și `googleapiclient.error`. Accesarea inbox-ului se realizează prin accesarea Google Cloud-ului și a oferi permisiuni pe contul care urmează sa fie analizat, este generat un token.json si un credentials.json care sunt folosite pentru permisiuni.

Odată ce accesul este disponibil urmează prima dată să fie analizate e-mailurile existente în inbox, fiecare e-mail are un id unic asociat, iar pentru a nu analiza de doua ori același mail este creat un fișier `verified_id.txt` unde acestea sunt stocate și verificate în timpul rulării. Fiecare e-mail are o structură JSON, astfel se iterează prin fiecare mesaj și se extrage conținutul ca în Listing 3.1:

```
content =
 ↪  service.users().messages().get(userId='me',id=message['id']).execute()
```

Listing 3.1. Extragerea conținutului unui e-mail

După extragerea conținutului unui e-mail, se iterează prin atașamente accesând părțile corespunzătoare din conținut : `content['payload']['parts']` și sunt salvate local, aceste atașamente sunt trimise pentru fiecare e-mail pe calea `http://localhost:9898/mail_content`, iar pentru a gestiona cazurile în care sunt mai multe atașamente de analizat se utilizează două cozi. O coadă este utilizată pentru a menține ordinea e-mail-urilor în cazul în care un e-mail conține măcar două atașamente, iar a doua coadă este utilizată pentru a gestiona rezultatul final. Pe ruta respectivă se realizează anumite prelucrări și apoi se trimit atașamentele pe rand către Cuckoo: `request.post("http://localhost:8090/tasks/create/file",files=file ,headers=HEADERS)`. După ce analizarea fișierelor a fost realizata de către Cuckoo se așteaptă rapoartele generate pe ruta `/tasks/done/<int:report_id>`. Pentru a demonstra funcționalitatea aplicației în cazurile în care trimiterea fișierelor executabile ar fi fost posibilă prin intermediul e-mail-ului ar urma procesarea rapoartelor de către modulele personale, urmând ca rezultatele sa fie trimise direct în inbox-ul clientului. Acest lucru se realizează conform diagramei 2.2.

Răspunsul returnat către utilizatorul de mail conține următoarele informații:

- Răspunsul modulului de potrivire cu reguli Yara
- Răspunsul modulului de clasificare
- Răspunsul modulului de clasificare cu ajutorul modulului de potrivire

După ce au fost primite rezultatele pentru un raport, inclusiv verdictul modulului Yara și cel al modulului de învățare automată, pentru a determina răspunsul celor două module se aplică următoarea logică: dacă fișierul analizat este considerat sigur din punct de vedere al modulului cu reguli Yara, se consideră că poate fi un ajutor pentru modulul de învățare automată. Pentru implementarea acestui aspect, se așteaptă mai întâi răspunsul de la modulul cu reguli care este stocat in variabila `response_rules`. Dacă răspunsul este *safe*, atunci variabila `model\_helper` devine zero pentru a semnala acest lucru. Apoi, se preia vectorul cu răspunsuri pentru modulul de învățare automată și se numără câte secvențe malițioase au fost detectate și câte secvențe inofensive au fost detectate. Decizia pentru răspunsul modulului de clasificare este similară cu un vot. Astfel, dacă sunt mai multe secvențe sigure, rezultatul oferit este *safe*, altfel este *dangerous*. După ce a fost stabilit verdictul pentru modulul de învățare automată si stocat in variabila

`response_model_no_help`, urmează evaluarea sa cu ajutorul modulului de reguli Yara. Mai întâi se verifică dacă variabila `model\_helper` este zero, caz în care modulul bazat pe reguli a semnalat că fișierul este sigur. Apoi, se incrementează numărul de voturi *safe* pentru modulul de învățare automată cu o singură valoare. Urmează reevaluarea prin vot, iar noul răspuns este salvat în variabila `response_model_help`. Toate cele trei răspunsuri sunt trimise către utilizatorul de e-mail inclusiv scorul oferit de Cuckoo care este preluat din raport.

3.2. Modulul Cuckoo

Pentru implementarea soluției, am adăugat funcționalitatea de notificare atunci când analiza unui fișier s-a finalizat. Pentru a dezvolta acest lucru, s-a modificat un fișier Cuckoo, care se află la calea specificată (`..cuckoo\core\scheduler.py`), prin adăugarea unei cereri PUT (Listing 3.2) care trimite ID-ul raportului care s-a terminat de analizat.

```
try:
  log.info("taskul e gata, sunt in scheduler.py")
  user_id = self.task.id
  url = "http://127.0.0.1:9898/tasks/done/"+str(user_id)
  requests.put(url)
  log.info(str(self.task.id))
except:
  log.info("Server for data processing is down.")
return True
```

Listing 3.2. Exemplu de intrare pentru TF-IDF api calls

3.3. Modulul MISP

Integrarea modulului MISP cu *sandbox*-ul Cuckoo este nativă, fiind necesară doar activarea într-un fișier de configurare pentru a asigura funcționalitatea de integrare a acestora. Codul de integrare necesar pentru realizarea acestei conexiuni este deja inclus și implementat de către autorii proiectului în cadrul *sandbox*-ului Cuckoo, neavând o contribuție personală la aceasta. Aceste precizări sunt necesare în acest capitol deoarece integrarea celor două module face parte din implementarea proiectului.

3.4. Modulul de extragere de caracterisitici

3.4.1. Extragere de caracterisitici pentru reguli Yara

Acest modul se ocupă de extragerea datelor din raportul JSON pentru a putea fi generate reguli Yara sau potrivite cu reguli Yara. Așadar această procesare este împărțita în două etape, prima etapă este cea prin care sunt extrase trăsăturile necesare, iar în a doua etapa se aplică algoritmii LCS și TF-IDF. Pentru a îndeplini obiectivul primei etape, caracteristicile de interes sunt stocate sub forma unui dicționar de dicționare, unde fiecare cheie reprezintă caracteristica de interes. Această modalitate este aleasă deoarece face preluarea datelor în etapa a doua mai facilă.

În vederea salvării informațiilor referitoare la traficul de internet, secțiunea `network` din raport este parcursă și informațiile corespunzătoare sunt stocate într-un dicționar. Cheia dicționarului reprezintă protocolul, iar valoarea reprezintă conținutul asociat. Dacă dicționarul nu este gol, acesta este adăugat în dicționarul final în câmpul `network` care urmează să fie returnat. Pentru a salva informațiile referitoare la apelurile API, acestea sunt extrase în două moduri diferite. Pentru aplicarea algoritmului LCS este nevoie de date care se accesează prin câmpul `processes` care este un vector de dicționare, unde fiecare dicționar conține mai multe informații despre un proces, inclusiv apeluri API și detaliile aferente acestora. Aceste detalii se regăsesc în câmpul `calls` din dicționar, care este și el reprezentat sub forma unui alt dicționar, astfel el conține detalii despre apeluri api cum ar fi categoria din care face parte, când a fost apelat, argumentele funcției dacă există

s.a. Deoarece se realizează doar analiza API-ului din dicționarul `calls`, câmpul de interes este accesat prin cheia "api". Astfel, pentru algoritmul LCS datele de interes sunt stocate în dicționarul final in câmpul "api_sequence". Pentru aplicarea algoritmului TF-IDF datele sunt extrase din câmpul `apistats` din raport, unde aceste informații sunt stocate într-un dicționar. Cheia dicționarului este reprezentată de numărul procesului, iar valoarea este un alt dicționar care conține perechi de forma <nume_apel_api>:<numar_apeluri>. La final pentru algoritmul TF-IDF datele acestea sunt stocate în dicționarul final in câmpul "apistats".

La finalul procesului de extragere a acestor date pentru prima etapa, se returnează un dicționar care conține cheile: "apistats", "network" și "api_sequence". În situația în care dicționarul nu conține nicio valoare, se returnează răspunsul "False". În caz contrar, se returnează dicționarul cu datele extrase.

În a doua etapă, dicționarul generat anterior este utilizat pentru a extrage trăsăturile relevante prin intermediul algoritmilor TF-IDF și LCS. Implementarea algoritmului TF-IDF a fost aplicată asupra apelurilor API, în timp ce pentru analiza traficului de internet s-a utilizat biblioteca `sklearn.feature_extraction.text.TfidfVectorizer`. Implementarea algoritmului TF-IDF pentru apeluri API se realizeaza preluand din dicționar valorile de la cheia `apistats` unde sunt stocate valorile ca în Figura 3.1.

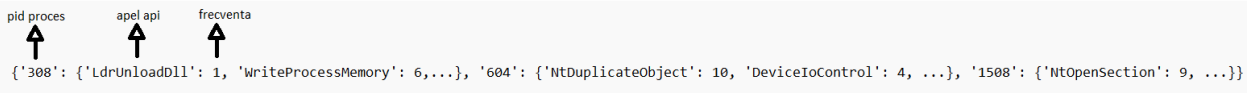

Figura 3.1. Exemplu de intrare pentru TF-IDF api calls/

Astfel, în primă fază, se extrag cheile (procesele) și se creează o matrice în care fiecare linie reprezintă aceste chei. Se aduna pentru fiecare proces numărul total de apeluri pentru a calcula TF, iar apoi pentru procesul respectiv coloanele sunt reprezentate de apelurile api corespunzătoare iar în matrice se completează valorile corespunzătoare pentru TF ca în Listing 3.3. Acest tip de accesare este posibil datorită utilizării `defaultdict(lambda: defaultdict(float))` pentru crearea matricei. Această abordare permite evitarea erorilor în cazul în care o cheie nu există, iar accesarea matricei se face similar cu accesarea unui dicționar de dicționare.

```
for process_nr in apistats.keys():
    total_words = sum(appearance for appearance in
     ↪  apistats.get(process_nr).values())
    for api in apistats.get(process_nr).keys():
        M[process_nr][api] = apistats.get(process_nr).get(api)/total_words
```

Listing 3.3. Calculare Term Frequency din TF-IDF.

Următorul pas constă în extragerea tuturor apelurilor API unice pentru a calcula IDF (Inverse Document Frequency) și a le stoca într-o listă separată. Apoi, această listă este parcursă în paralel cu matricea creată anterior, cu scopul de a număra numărul de procese în care apare fiecare apel API. După procesarea menționată, se poate calcula IDF (Inverse Document Frequency) conform formulei descrise în capitolul 2. Pentru fiecare apel API, valoarea IDF este stocată într-un dicționar, unde cheia este reprezentată de apelul API în cauză. Ulterior se parcurge matricea cu valorile TF și se înmulțește cu valoarea IDF corespunzătoare pentru a obtine TF-IDF. La final se selectează primele trei apeluri cu cea mai mare valoare.

Pentru aplicarea algoritmului TF-IDF peste traficul de internet datele trebuie procesate astfel încât utilizarea bibliotecii sa fie posibilă. Așadar, datele de intrare sunt separate în ip-uri și dns-uri, unde pentru ip-uri se creează o lista care sa conțină pe fiecare poziție ip-urile pentru un anumit protocol (astfel o poziție reprezinta un document pentru algoritm), iar pentru dns se procedează la fel. După prelucrarea respectivă, se utilizează funcția `TfidfVectorizer()` în Python,

care primește ca parametru modalitatea de separare a cuvintelor. În mod implicit, regex-ul utilizat este configurat pentru a separa cuvintele bazându-se pe punctuație. Pentru a permite separarea și identificarea IP-urilor și DNS-urilor în mod distinct, regex-ul a fost modificat.

După etapa menționată, sunt selectate primele cinci adrese IP cu scorul TF-IDF cel mai mare, precum și primele cinci adrese DNS cu scorul TF-IDF cel mai mare.

După calcularea primelor doua trăsături urmează calcularea LCS-ului pentru a găsi cea mai lunga secventa comuna dintre două procese. Procesele care intră în aceasta analiza sunt primele două care prezintă o secventa api, astfel mai întâi se verifica dimensiunea acestora pentru a vedea dacă este posibila stocarea lor. Dacă nu este posibilă stocarea lor se returnează mesajul `'Not enough memory'`, altfel se creează o matrice de dimensiunea secventelor plus unu, cu prima linie și prima coloana zerorizate. Fiecare linie reprezintă un apel API din secvența primului proces, iar fiecare coloană reprezintă un apel API din secvența celui de-al doilea proces. Se parcurge matricea, iar când valoarea unei linii corespunde cu cea de pe o coloană (avem un cuvânt comun) se incrementează cu unu valoarea de pe diagonală sa (acest lucru se întâmplă deoarece pe diagonală se afla lungimea secventei comune pana atunci). Se continuă parcurgerea și elementele preiau maximul dintre valoarea liniei anterioare și a coloanelor anterioare. Matricea este umplută în conformitate cu descrierea dată, iar la final, lungimea celei mai lungi secvențe comune este determinată de valoarea din colțul din dreapta jos al matricei. Această valoare reprezintă lungimea maximă a unei secvențe comune între cele două procese. După ce matricea a fost completată ca în 3.1 urmează parcurgerea ei pentru a putea găsi secvența de api-uri comune. Pentru a realiza acest lucru, se parcurge matricea începând din colțul dreapta-jos ca în Figura 3.2. Apoi, se reconstituie drumul în următorul mod: dacă valoarea elementului din linia curentă și coloana curentă este egală, înseamnă că un API din secvență a fost găsit, și acesta este adăugat la rezultat. În acest caz, indicele se modifică astfel încât să se deplaseze pe diagonala din stânga. În caz contrar, dacă valorile din coloana anterioară și linia anterioară sunt egale, indicele se modifică astfel încât să se deplaseze cu un rând în sus. În situația în care valorile difera, indicele se modifică astfel încât să se îndrepte către valoarea maximă disponibilă. La final șirul rezultat este inversat.

După toate aceste procesări se returnează rezultatul sub forma unui dicționar de forma `{"API":API_result,"LCS":LCS_result,"NETWORK":NETWORK_result}` , în cazul în care nu se găsesc caracterisitici se returnează răspunsul FALSE. În faza de antrenare aceste rezultate sunt stocate în Redis pentru a putea fi preluate ușor de către modulul de generare de reguli.

Tabelul 3.1. Matrice pentru determinarea lungimii secvenței

		LdrGetDllHandle	NtClose	LdrGetProcedureAddress	NtClose	LdrGetProcedureAddress
	0	0	0	0	0	0
LdrGetDllHandle	0	1	1	1	1	1
LdrGetProcedureAddress	0	1	1	2	2	2
LdrGetProcedureAddress	0	1	1	2	2	3
LdrGetProcedureAddress	0	1	1	2	2	3

```
Rezultat: ['LdrGetDllHandle', 'LdrGetProcedureAddress',
 ↪  'LdrGetProcedureAddress']
```

		LdrGetDllHandle	NtClose	LdrGetProcedureAddress	NtClose	LdrGetProcedureAddress
	0	0	0	0	0	0
LdrGetDllHandle	0	1	1	1	1	1
LdrGetProcedureAddress	0	1	1	2	2	2
LdrGetProcedureAddress	0	1	1	2	2	3
LdrGetProcedureAddress	0	1	1	2	2	3

Figura 3.2. Modalitate de traversare a matricei pentru găsirea celei mai lungi secvențe comune

3.4.2. Extragere de caracterisitici pentru model

Extragerea caracteristicilor pentru Random Forest se efectuează într-un mod similar cu extragerea trăsăturilor pentru LCS. Cu toate acestea, există o diferență importantă în ceea ce privește selecția datelor. În cazul Random Forest, se extrag doar apelurile API din procesele care sunt direct copii ale procesului părinte.

Procesul de extragere a caracteristicilor constă în două etape distincte. În prima etapă, se identifică procesele care sunt copii direcți ai procesului părinte iar in a doua etapa se selectează secvența de apeluri API de lungime o sută din apelurile non-consecutive ale acestor procese identificate anterior.

În prima etapă a procesului de selectare a datelor, se accesează câmpul `processtree` din secțiunea `behavior` a raportului. Acest câmp este reprezentat de un vector de dicționare, unde fiecare dicționar conține informații despre un proces. Pentru a identifica procesele care sunt copii direcți ai procesului părinte, se extrage cheia relevantă, în acest caz "ppid" (identificatorul procesului), și se stochează valorile acestor chei într-o listă. În a doua etapă a procesului de selecție a datelor, se parcurge secțiunea `behavior/processes` descrisă în modulul de extragere a datelor pentru generarea regulilor Yara. În această etapă, mai întâi se verifică dacă PPID-ul procesului respectiv se găsește în lista proceselor părinte obținută în etapa anterioară. Dacă PID-ul procesului are un PPID (PID-ul părintelui) prezent în lista proceselor părinte, atunci se începe parcurgerea dicționarului `calls` asociat acestui proces. În cadrul acestui dicționar, se extrag apelurile API și se introduc într-o altă listă, care va conține doar secvențele de API de orice lungime care aparțin unui proces părinte. Odată ce este formată această lista, ea este parcursă și sunt selectate doar api-urile care nu sunt consecutive conform exemplului prezentat în capitolul 2. Dacă o secvență de API are lungimea de o sută, ea este introdusă într-o altă listă finală, care va conține toate secvențele de API non-consecutive cu această lungime. Această listă finală este apoi returnată ca rezultat al procesului de extragere a caracteristicilor pentru algoritmul Random Forest.

3.5. Modulul de antrenare și evaluare al modelelor

Pentru antrenarea și evaluarea modelelor este utilizată biblioteca `sklearn` din Python prin care sunt obținute rezultatele din tabelul. Înainte de a selecta modelul final care va fi utilizat în lucrarea de diplomă este necesară evaluarea modelelor menționate în 2.6. Această evaluare este realizată utilizând funcția `sklearn.model_selection.GridSearchCV`, care permite explorarea și optimizarea parametrilor modelelor prin căutarea în grilă a combinațiilor optime. Prin această abordare, se poate selecta modelul cu cea mai bună performanță și capacitate de generalizare pentru problema specifică tratată în lucrare.

Pentru a evalua cele patru modele propuse, s-au explorat diferiți parametri în cadrul procesului de optimizare, astfel că au fost încercate următoarele combinații de valori pentru `GridSearchCV`:

1. SVM:

```
param_grid = [{'C': [1, 10, 100, 1000], 'kernel': ['linear']},
      {'C': [1, 10, 100, 1000], 'kernel': ['rbf'], 'gamma': [0.01,
       ↪  0.05, 0.5, 1, 3]},
```

```
        {'C': [1, 10, 100, 1000], 'kernel': ['poly'], 'degree':
         ↪  [0.5,1,2, 3, 4], 'gamma': [0.01, 0.05, 0.5, 1, 3]}
        ]
```

2. LR:

```
param_grid = [
      {'penalty': ['l1', 'l2', 'elasticnet'],
       'C': [0.001,0.01,0.1,1,10,100,1000],
       'solver': ['lbfgs', 'newton-cg', 'liblinear', 'sag', 'saga'],
       'max_iter': [100, 1000, 2500, 5000]
       }
  ]
```

3. KNN:

```
param_grid = {'n_neighbors': [int(x) for x in range(60)],
          'weights': ['uniform', 'distance'],
          'metric': ['cosine', 'euclidean', 'manhattan']}
```

4. RF:

```
param_grid = {
    'bootstrap': [True,False],
    'max_depth': [10, 15,20,25,30],
    'max_features': ['auto','sqrt','log2'],
    'min_samples_leaf': [1,2,3  , 4, 5, 6,7,8,9,10]
    'n_estimators': [int(x) for x in range(50,2000,100)],
    'criterion':['gini','entropy','log_loss']
}
```

După determinarea parametrilor folosind această abordare, s-a calculat pentru fiecare model *precision, recall, f1-score, support* utilizând funcția `sklearn.metrics.classification_report`. Pe baza acestor rezultate, a fost selectat modelul cu cele mai bune performanțe, care în acest caz este Random Forest.

Implementarea modelelor este făcută prin biblioteca `sklearn`, iar despărțirea datelor e realizata prin `train_test_split` în 25% date de test iar 75% date de antrenare.

3.6. Modulul de generare a regulilor Yara

În faza de antrenare, după prelucrarea datelor conform secțiunii 3.4.1, este necesară generarea regulilor Yara pentru a putea potrivi ulterior aceste reguli cu intrări noi. Datele procesate sunt preluate din Redis și apoi convertite în reguli utilizând clasa `Rule`, unde constructorul are forma din Listing 3.4, `type` reprezintă scorul oferit de Cuckoo, feature specifică dacă trăsătura este LCS, API_CALL sau NETWORK, acest lucru este necesar deoarece diferă procesările care trebuie făcute, `description` este numele fișierului care a generat respectivele caracteristici iar api reprezinta valoarea din dicționar (LCS, API_CALL sau NETWORK).

```
def __init__(self, type, feature, description,api):
    self.type = type
    self.feature = feature
    self.description = description
    self.api = api
```

Listing 3.4. Constructorul clasei Rule pentru convertire in reguli

După instanțierea corectă a unui obiect `Rule`, este apelată funcția `api_formatter` pentru a crea forma regulii Yara utilizând informațiile primite în constructor. Cazul caracteristicii de

LCS, necesită o atenție suplimentară datorită posibilității întâlnirii unei erori cauzate de lungimea unei secvențe. Pentru a evita astfel de erori, se efectuează două procesări suplimentare. În primul rând, secvența este codificată conform capitolului 2, apelând funcția `sequence_fitter`. Această funcție codifică apelurile utilizând fișierul `api.json`, care a fost generat prin intermediul unui script personalizat și conține toate apelurile API întâlnite în setul de antrenament sub forma unui dicționar, unde fiecare nume de API este asociat cu o codificare specifică. În al doilea rând, după această prelucrare, în cazul secvențelor LCS, se utilizează un regex care permite potrivirea unei secvențe cu o regulă LCS chiar și atunci când secvența respectivă este mai lungă și conține regula.

După formatarea regulilor, acestea sunt salvate în fișiere `.yar`. Fiecare caracteristică este salvată în fișierul corespunzător, respectiv `LCS.yar`, `API.yar` și `NETWORK.yar`. Această salvare în fișiere separate permite ulterior potrivirea lor individuală și utilizarea lor. Pentru a le salva în formatul unei reguli Yara a fost necesară suprascrierea metodei `str` ca în Listing 3.5.

```
def __str__(self):
  return "rule "   +  str(self.feature) + "_" +
  self.description + "\n"  + "{" +
  "\n\tmeta:\n\t\ttype = \""+ self.type  +
  "\"\n\t" + "strings:\n" + self.api +
  "\n\tcondition:\n\t\t" +
  self.condition+"\n}"
```

Listing 3.5. Modalitate de convertire in formatul Yara

După această etapă, în faza de antrenare, se creează reguli Yara care sunt utilizate ulterior pentru a potrivi caracteristici din fișiere noi cu aceste reguli. Regulile Yara sunt create pe baza datelor de antrenament, acestea sunt apoi aplicate pentru a identifica și clasifica caracteristici similare în fișierele noi.

3.7. Modulul de potrivire cu reguli Yara

Modulul de potrivire cu reguli Yara este utilizat exclusiv după finalizarea fazei de antrenare și în momentul în care aplicația este activă. După ce regulile Yara au fost create și stocate într-un format adecvat în timpul fazei de antrenare, acestea sunt utilizate în mod activ în timpul funcționării aplicației dacă exista caracteristicele corespunzătoare.

Este important de menționat că faza de antrenare și crearea regulilor Yara este distinctă de procesul de potrivire în timp real. Faza de antrenare este responsabilă cu generarea regulilor Yara pe baza setului de date de antrenare, în timp ce modulul de potrivire este utilizat pentru a aplica aceste reguli pe date noi în timpul funcționării aplicației.

Pentru a realiza aceasta potrivire este utilizată biblioteca `yara-python`. Odată ce modulul primește caracteristicile potrivite ale fișierului analizat, acestea sunt stocate în fișiere separate pentru utilizare ulterioară (așa cum este exemplificat în Figura 3.3). Pentru potrivirea regulilor de LCS este necesară codificarea ei ca în 3.6. Acest proces de potrivire Yara implică compararea unui fișier JSON cu un fișier YAR care conține regulile corespunzătoare.

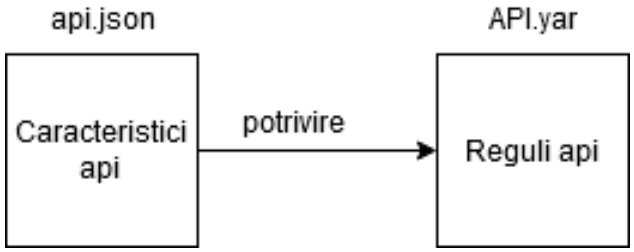

Figura 3.3. Modalitate de potrivire cu reguli Yara pentru trăsătura de TF-IDF aplicat peste apeluri API.

Acest lucru este realizat prin folosirea metodei *match()* ca în Listing 3.6, însa se face pentru fiecare caracteristica.

```
        rules_api = yara.compile(filepath='Yara_matcher/API.yar')
matches = rules_api.match('api.json', callback=mycallback,
 ↪  which_callbacks=yara.CALLBACK_MATCHES)
```

Listing 3.6. Potrivire cu reguli Yara pentru o caracteristica

Pentru a putea lua o decizie pentru a clasifica fișierul în categoria de malware sau goodware, se efectuează o numărătoare a regulilor malițioase și a regulilor inofensive cu care fișierul se potrivește. Implementarea acestei logici se face prin funcția `mycallback` care este apelată de fiecare dată când se face o potrivire cu o regula. La final se compara valoarea din câmpul `benign_rules` cu câmpul `malware_rules` iar în funcție de rezultat se returnează răspunsul *safe* sau *dangerous*.

3.8. Modulul de clasificare

Modulul de clasificare utilizează algoritmul Random Forest pentru a efectua clasificarea fișierelor în categoriile de malware sau goodware. Acesta funcționează preluând trăsăturile procesate conform 3.4.2 și sunt furnizate modelului. Modelul antrenat este încărcat prin biblioteca `joblib` iar ulterior datele noi sunt furnizate către model ca în Listing 3.7.

```
def rf(new_data):
    r1 = []
    for key in new_data:
        for api_seq in new_data[key]:
            r1.append(int(rf_balanced.predict(api_seq)[0]))
    rezultat = {"balanced":r1}
    return jsonify({"response":rezultat})
```

Listing 3.7. Furnizare date noi către model

Datele furnizate sunt sub forma unui dicționar unde cheia reprezinta procesul iar valoarea este secventa de o suta de apeluri api non-consecutive. Fiecare secvență este procesată de către model, iar rezultatul este stocat într-un dicționar, unde valoarea este o listă ce conține toate ieșirile modelului pentru fiecare secvență în parte. Dacă un fișier conține o singură secvență, atunci dicționarul va avea o valoare de tip listă cu lungimea unu. În cazul în care fișierul conține mai multe secvențe, dicționarul va avea o valoare de tip listă cu lungimea corespunzătoare numărului de secvențe.

Capitolul 4. Testarea aplicației și rezultate experimentale

4.1. Punerea în funcțiune a aplicației

Pentru a utiliza soluția, trebuie să pornim modulele corespunzătoare și să le configurăm pe anumite porturi. În mod specific, trebuie să pornim *sandbox*-ul Cuckoo, modulul de extragere de e-mail-uri, modulul de procesare a unui e-mail și modulul de procesare a rapoartelor generate de Cuckoo. Modulul MISP nu trebuie pornit explicit, e suficientă doar integrarea cu *sandbox*-ul. Aceste module sunt configurate pe următoarele porturi pentru a permite comunicarea și interoperabilitatea între ele:

- Cuckoo – port:8080
- MISP – port:34732 (redirectat)
- Procesare de e-mail – port:9898
- Procesare rapoarte – port:10101
- Redis – port:6379

Pentru a porni *sandbox*-ul sunt mai multe etape, mai întâi trebuie configurată rețeaua pentru Cuckoo prin comenzile din 4.1.

```
sudo sysctl -w net.ipv4.conf.vboxnet0.forwarding=1
sudo sysctl -w net.ipv4.conf.enp0s3.forwarding=1

sudo iptables -t nat -A POSTROUTING -o enp0s3 -s 192.168.56.0/24 -j
 ↪  MASQUERADE
sudo iptables -P FORWARD DROP
sudo iptables -A FORWARD -m state --state RELATED,ESTABLISHED -j ACCEPT
sudo iptables -A FORWARD -s 192.168.56.0/24 -j ACCEPT
```

Listing 4.1. Comenzi configurare rețea Cuckoo

Apoi se pornește rutarea prin comanda `cuckoo rooter --sudo --group cuckoo` (pentru a rula această comandă în fișierul `routing.conf` trebuie modificate corespunzător anumite variabile) urmând să fie pornit *sandbox*-ul prin comanda `cuckoo`, interfața web prin:

```
cuckoo web --host 127.0.0.1 --port 8080
```

și ulterior se pornește api-ul Cuckoo prin `cuckoo api`.

Pentru utilizarea contului de Gmail este nevoie de cele două fișiere `credentials.json` care conțin informații despre conturile de e-mail care sunt utilizate în cadrul proiectului de diplomă, și trebuie regenerate fișierele `token.json` în cazul în care acestea au expirat.

Accesarea modulului MISP se face prin navigarea către URL-ul:
`https://localhost/users/login` pentru a putea vizualiza ulterior corelări și alte informații despre fișierele analizate.

4.2. Rezultate experimentale

4.2.1. Rezultate modul Yara

Scenariile de test sunt realizate separat pentru cele două module de evaluare a fișierelor. Astfel, modulul de generare de reguli Yara este antrenat și testat pe setul de date menționat în subcapitolul 2.8. Fișierele malware care sunt în setul respectiv de date sunt patru variante populare de ransomware: WannaCry, Cerber, Crysis și Hydracrypt.

WannaCry - apărut în 2017, acest crypto ransomware preia controlul asupra datelor și solicită o sumă de bani în Bitcoin pentru a returna accesul la ele și are ca țintă calculatoarele care rulează sistemul de operare Microsoft Windows. Acest malware a profitat de *exploit*-ul *EternalBlue* dezvoltat de Agenția de Securitate Națională a Statelor Unite care avea o vulnerabilitate pe protocolul *Server Message Block (SMB)*.

Cerber - apărut în 2016, acest ransomware utilizează un model de Ransomware-as-a-service (RaaS), în care afiliații cumpără și ulterior răspândesc malware-ul. Principala metodă de distribuire a acestui malware este prin intermediul phishing-ului prin e-mail. Acest software rău intenționat verifică automat locația dispozitivului utilizatorului și se instalează doar dacă acesta nu se află în Rusia sau într-unul dintre statele vecine (Statele postsovietice). În general e-mailul conținea un atașament sub forma unui document Word care avea macro-uri malițioase pentru a instala software-ul și a cripta datele utilizatorului.

Crysis - detectat pentru prima dată în 2016, Crysis se strecoară în computerul utilizatorului prin intermediul e-mailurilor care conțin atașamente cu dublă extensie de fișier pentru a părea non-executabile. Crysis are capacitatea de a cripta peste 185 de tipuri de fișiere pe unități de stocare fixe și detașabile (de exemplu, USB-uri și discuri externe), precum și partajări de rețea, utilizând o combinație de algoritmi de criptare RSA și AES. Pentru a asigura infectarea, Crysis șterge copiile de rezervă ale sistemului, care servesc drept copii de siguranță ale fișierelor sau volumelor computerului. Ca măsură de persistență, ransomware-ul creează și introduce noi valori în Registrul de sistem Windows. Aceasta permite malware-ului să ruleze de fiecare dată când utilizatorul se autentifică în sistem, ceea ce face mai dificilă eliminarea sa. Datele criptate sunt marcate cu o extensie *.crysis* în numele fișierelor.

Hydracrypt - apărut în 2016, acest malware adună anumite informații despre mașina țintă și le comunică către un C2 (*Command & Control*). HYDRACRYPT este un ransomware care criptează majoritatea fișierelor stocate pe calculatoarele infectate și adaugă extensia 'hydracrypt_ID_[8 caractere aleatoare]' la toate fișierele criptate. După criptare, HYDRACRYPT afișează un mesaj în care se cere victimelor să achiziționeze un software (dezvoltat de către infractorul cibernetic) pentru a decripta fișierele. Mesajul mai precizează că victimele trebuie să plătească în termen de 72 de ore, altfel cheia privată (folosită pentru decriptarea fișierelor) va fi distrusă și fișierele vor rămâne criptate pentru totdeauna.

Aceste variante de ransomware utilizează diferite metode de infectare, ceea ce duce la crearea unor reguli Yara mai diverse, dar totuși încadrate în același domeniu. Pentru setul de date de fișiere inofensive (menționat în subcapitolul 2.8) sunt folosite diverse executabile Windows.

Pentru generarea regulilor (antrenare) s-au folosit în total 370 de fișiere dintre care 174 inofensive și 196 de malware-uri. Din toate aceste date doar 328 (174 inofensive și 154 periculoase) au caracteristicele necesare pentru a genera cel puțin o regulă. Pentru testarea regulilor s-au folosit în total 73 de fișiere dintre care 37 inofensive și 36 de malware-uri.Din toate aceste date doar 61 (37 inofensive și 25 periculoase) au caracteristicele necesare pentru a genera cel puțin o regulă. Astfel din datele inițiale de 443 de fișiere, 83% sunt date de antrenare și 17% de testare. Din datele de antrenare 53% sunt inofensive și 47% sunt malware iar din datele de testare 58% sunt inofensive și 42% sunt periculoase. S-a încercat păstrarea unui raport echilibrate între datele utilizate însă partea dificilă constă în a găsi date care sa conțină cel puțin o caracteristică dorită.

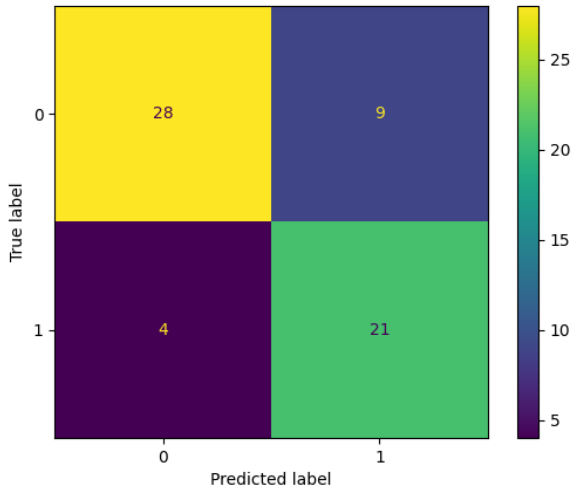

Figura 4.1. Matrice de confuzie pentru modulul de potrivire cu reguli Yara.

Se obțin următoarele valori:

Tabelul 4.1. Raport de clasificare generat de sklearn pentru modulul de clasificare după reguli Yara

	Precision	**Recall**	**F1-score**	**Support**
0	0.88	0.76	0.81	37
1	0.70	0.84	0.76	25
accuracy	0.79	0.79	0.79	0.79

Rezultate modul învățare automată

Pentru modulul de învățare automată s-a utilizat un set de date popular pentru analiza de API-uri [15], acest set de date nu este foarte echilibrat (1079 fișiere sigure, 42 797 fișiere malware), astfel a fost împărțit în 1079 de fișiere sigure și 1086 de fișiere malware, la care s-au adăugat datele modulului Yara, având în total 2535 de date. Setul de date este împărțit 25% date de test și 75% date de antrenare, astfel am obținut pentru cei patru algoritmi următoarele rezultate:

[15] https://ieee-dataport.org/open-access/malware-analysis-datasets-api-call-sequences

Random Forest

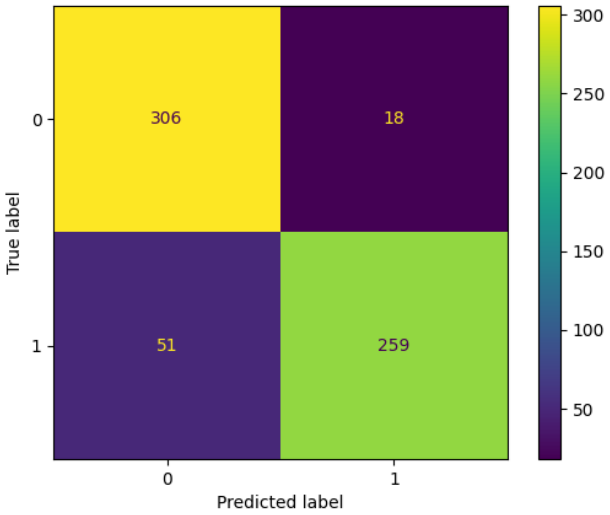

Figura 4.2. Matrice de confuzie pentru Random Forest.

Cu următoarele valori:

Tabelul 4.2. Raport de clasificare generat de sklearn pentru Random Forest

	Precision	**Recall**	**F1-score**	**Support**
0	0.866856	0.944444	0.903988	324.000000
1	0.935943	0.848387	0.890017	310.000000
accuracy	0.897476	0.897476	0.897476	0.897476

KNN

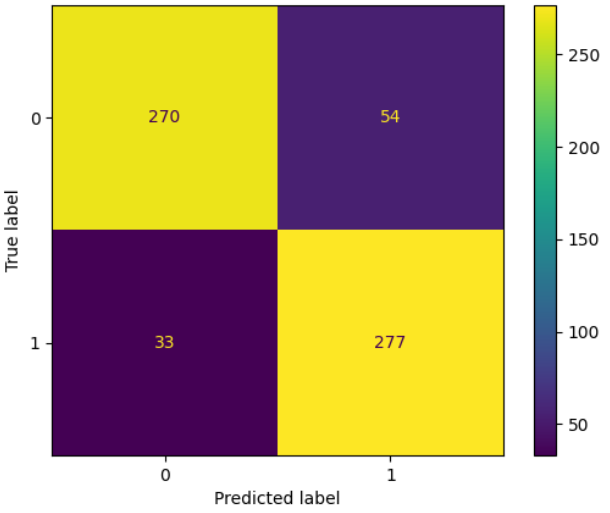

Figura 4.3. Matrice de confuzie pentru KNN.

Cu următoarele valori:

Tabelul 4.3. Raport de clasificare generat de sklearn pentru KNN

	Precision	Recall	F1-score	Support
0	0.891089	0.833333	0.861244	324.000000
1	0.836858	0.893548	0.864275	310.000000
accuracy	0.862776	0.862776	0.862776	0.862776

LR

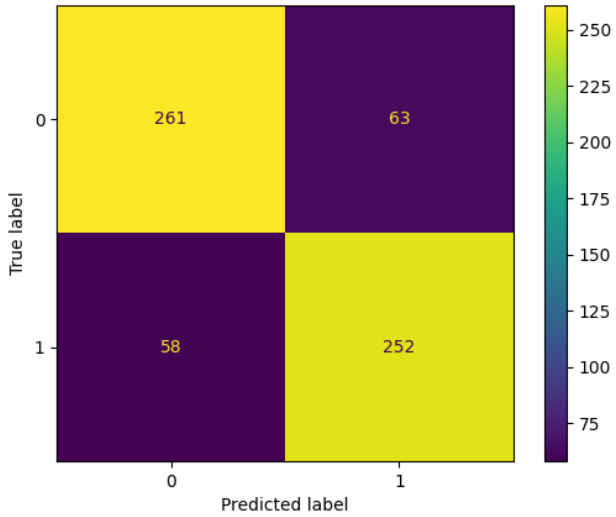

Figura 4.4. Matrice de confuzie pentru LR.

Cu următoarele valori:

Tabelul 4.4. Raport de clasificare generat de sklearn pentru Regresia Logistică

	Precision	Recall	F1-score	Support
0	0.818182	0.805556	0.811820	324.000000
1	0.800000	0.812903	0.806400	310.000000
accuracy	0.809148	0.809148	0.809148	0.809148

SVM

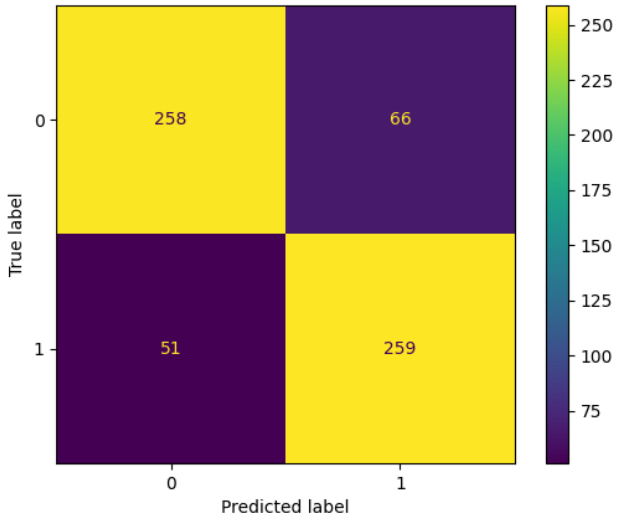

Figura 4.5. Matrice de confuzie pentru SVM.

Cu următoarele valori:

Tabelul 4.5. Raport de clasificare generat de sklearn pentru SVM

	Precision	Recall	F1-score	Support
0	0.834951	0.796296	0.815166	324.000000
1	0.796923	0.835484	0.815748	310.000000
accuracy	0.815457	0.815457	0.815457	0.815457

Pe baza rezultatelor din tabelele obținute din Figura 4.2, Figura 4.3 , Figura 4.4 și Figura 4.5, s-a decis să se utilizeze modelul Random Forest în cadrul proiectului de diplomă. Deși cele două module utilizează seturi de date distincte și nu pot fi comparate, ele își propun să lucreze în sinergie pentru a oferi o analiză mai completă și precisă. Modulul bazat pe Random Forest are rolul de a fi un sprijin în procesul de identificare al malware-urilor. Prin integrarea acestui modul în arhitectura generală a aplicației, se urmărește îmbunătățirea soluției propuse.

4.2.2. Rezultate Cuckoo

Pentru a testa funcționalitatea modulului de Gmail este creat un cont de test `licenta.stefanparaschiv@gmail.com` care conține e-mail-uri cu un atașament, mai multe atașamente sau niciun atașament. Cum trimiterea fișierelor executabile nu este posibilă, în Figura 4.6 se poate observa răspunsul oferit pentru analizarea unui PDF care nu este infectat.

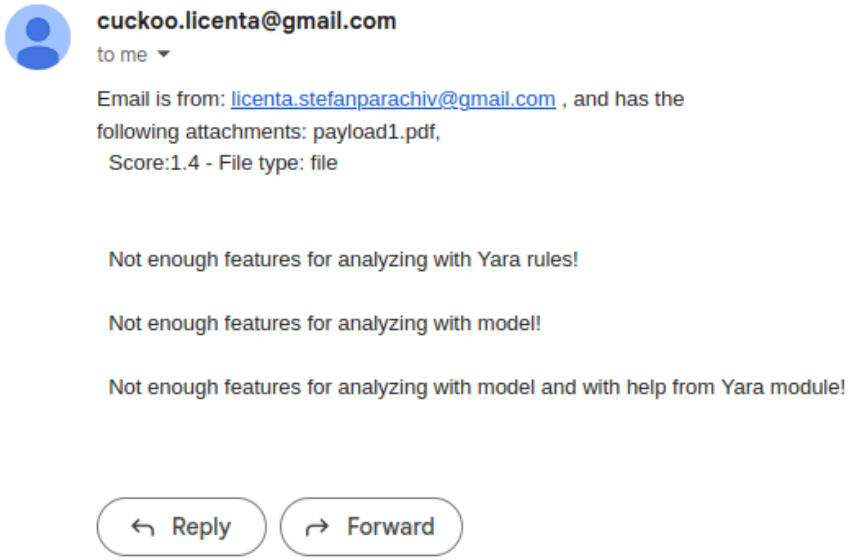

Figura 4.6. Exemplu de răspuns pentru un PDF.

Având în vedere că fișierul este în format PDF, nu dispune de foarte multe caracteristici necesare pentru analiza propusă de cele două module. Dacă ar fi fost un fișier executabil, pentru fiecare caz ar fi fost indicat dacă este considerat *safe* sau *dangerous*. Cu toate acestea, în continuare se pot obține informații utile pentru a sprijini procesul de analiză și luare a deciziilor chiar si dacă nu sunt suficiente trăsături pentru analiză.

4.3. Cazuri de utilizare a aplicației

Aplicația propusă oferă o soluție pentru identificarea și analizarea malware-urilor în diferite contexte. Prin utilizarea analizei comportamentale și integrarea mai multor soluții existente, aplicația este capabilă să detecteze și să analizeze în timp real activitățile suspecte ale malware-urilor. Aceasta poate fi implementată în cadrul organizațiilor pentru a proteja infrastructura lor de potențiale amenințări, necesitând doar adaptarea modulului de extragere de e-mail-uri pentru a se potrivi cu domeniul organizației respective. De asemenea, aplicația poate fi folosită și de către persoane individuale care doresc să beneficieze de un nivel suplimentar de protecție. Prin adoptarea unei arhitecturi modulare și integrarea cu platforme precum MISP, aplicația asigură o înțelegere mai profundă a fișierelor suspecte.

În scenariul de test al aplicației modulele au fost evaluate separat, modulul Yara a obținut o acuratețe de 79% iar modulul de clasificare 89.7% astfel se consideră ca sistemul este pregătit să analizeze fișiere noi. Luând în considerare faptul că Gmail nu permite trimiterea fișierelor executabile, se pot livra fișiere manual către *sandbox* pentru a vedea care sunt rezultatele celor două module împreună.

Concluzii

În cadrul proiectului de diplomă s-a dezvoltat o aplicație care se concentrează pe analiza fișierelor din *inbox*-ul unui utilizator într-un mediu de *sandbox*.Aplicația utilizează două module distincte pentru a determina natura și caracteristicile fișierelor analizate pe baza raportului generat de *sandbox*. Primul modul, bazat pe reguli Yara, compară anumite atribute ale fișierului cu regulile predefinite generate în faza de antrenare pentru a lua o decizie. Al doilea modul, bazat pe învățare automată, antrenează un model pe baza unor atribute relevante și furnizează un rezultat de clasificare. Această abordare modulară permite o analiză cuprinzătoare și flexibilă, furnizând utilizatorilor informații importante despre natura și potențialul periculos al fișierelor.

Aplicația este integrată cu platforma MISP (Malware Information Sharing Platform), permițând corelarea evenimentelor personale cu alte evenimente existente. Această integrare aduce un avantaj semnificativ, deoarece oferă utilizatorilor posibilitatea de a obține o imagine de ansamblu asupra amenințărilor de securitate și de a identifica conexiuni, modele și atribute care pot fi relevante în contextul analizei malware-urilor.

Deși cele două module folosesc ca atribute principale apeluri API, ele sunt procesate diferit. Modulul Yara utilizează algoritmi precum Longest Common Subsequence (LCS) și Term Frequency-Inverse Document Frequency (TF-IDF) pentru procesarea datelor și are ca atribut suplimentar traficul de internet. Algoritmul LCS identifică cea mai lungă secvență comună de apeluri de sistem a unui fișier, în timp ce algoritmul TF-IDF evaluează relevanța apelurilor API, a adreselor ip și a DNS-urilor. Pe de altă parte, modulul de învățare automată selectează secvențe de apeluri API de lungime o sută care nu sunt consecutive și, după compararea a patru modele (Random Forest, Logistic Regression, Support Vector Machine și K Nearest Neighbor), s-a ales modulul Random Forest ca model final pentru acest proiect.

În concluzie, proiectul de diplomă este o aplicație care utilizează două module distincte pentru analiza fișierelor din *inbox*-ul utilizatorului într-un mediu de *sandbox*. Modulul *sandbox* asigură un mediu controlat pentru analiza comportamentului fișierelor, în timp ce integrarea cu Gmail facilitează accesul și analiza automată a fișierelor din inbox. Integrarea cu platforma MISP are avantajul corelării evenimentelor personale cu alte evenimente existente, furnizând utilizatorilor o perspectivă mai amplă asupra amenințărilor de securitate. Astfel, această soluție reprezintă un instrument de sprijin pentru alte soluții, pentru protejarea infrastructurii și a datelor utilizatorilor împotriva potențialelor atacuri și vulnerabilități.

Direcții viitoare de dezvoltare

Aplicația poate fi îmbunătățită prin adăugarea de noi module și tehnici de analiză pentru a detecta și evalua o gamă mai largă de amenințări și comportamente suspecte. Aceasta poate include adăugarea de algoritmi de învățare automată avansată care reprezintă o metodă de detecție comportamentală mai sofisticată. Un exemplu concret de direcție viitoare pentru dezvoltarea aplicației ar putea fi implementarea unui modul bazat pe algoritmul *n-gram*. Acest modul ar extrage API-urile relevante din comportamentul fișierelor analizate, permițând utilizarea lor în diverse algoritmi de clasificare sau generarea de reguli personalizate pentru completarea regulilor Yara.

O altă direcție importantă de dezvoltare ar consta în analizarea detaliată a diferitelor atribute din raportul generat de Cuckoo Sandbox. Un exemplu concret ar putea fi realizarea unei analize aprofundate a apelurilor API, luând în considerare atât numele API-ului utilizat, cât și parametrii pe care acestea îi primesc în funcție. De asemenea se poate realiza o analiză mai amănunțită a traficului de internet, eventual să se realizeze și integrarea Cuckoo Sandbox cu IDS Suricata.

O direcție mai complexă necesită o mapare pe tactici de apeluri API după clasificare pe *framework*-ul MITRE ATT&CK® astfel încât să se poată determina tactica de operare a malware-

ului

Un avantaj semnificativ al arhitecturii propuse este ușurința de integrare a modulelor adiționale. Pe măsură ce apar idei noi și tehnologii avansate, acestea pot fi implementate în sistem cu relativă ușurință, fără a necesita modificări majore în arhitectura existentă.

Bibliografie

[1] T. H. News, "How to build a custom malware analysis sandbox," https://thehackernews.com/2022/03/how-to-build-custom-malware-analysis.html, 2022, Ultima accesare: 11.03.2023.

[2] Cisco, "What is malware?" https://www.cisco.com/site/us/en/products/security/what-is-malware.html, 2023, Ultima accesare: 11.03.2023.

[3] K. Baker, "What is malware?" [Online]. Available: https://www.crowdstrike.com/cybersecurity-101/malware/

[4] J. Hwang, J. Kim, S. Lee, and K. Kim, "Two-stage ransomware detection using dynamic analysis and machine learning techniques," *Wireless Personal Communications*, vol. 112, pp. 2597–2609, 2020.

[5] C. Wang, J. Pang, R. Zhao, and X. Liu, "Using api sequence and bayes algorithm to detect suspicious behavior," in *2009 International Conference on Communication Software and Networks*. IEEE, 2009, pp. 544–548.

[6] M. Alazab, S. Venkatraman, P. A. Watters, M. Alazab *et al.*, "Zero-day malware detection based on supervised learning algorithms of api call signatures." *AusDM*, vol. 11, pp. 171–182, 2011.

[7] M. I. Sharif, V. Yegneswaran, H. Saidi, P. A. Porras, and W. Lee, "Eureka: A framework for enabling static malware analysis." in *ESORICS*, vol. 8. Springer, 2008, pp. 481–500.

[8] Y. Ki, E. Kim, and H. K. Kim, "A novel approach to detect malware based on api call sequence analysis," *International Journal of Distributed Sensor Networks*, vol. 11, no. 6, p. 659101, 2015.

[9] M. Alazab, S. Venkataraman, and P. Watters, "Towards understanding malware behaviour by the extraction of api calls," in *2010 second cybercrime and trustworthy computing workshop*. IEEE, 2010, pp. 52–59.

[10] C. Ravi and R. Manoharan, "Malware detection using windows api sequence and machine learning," *International Journal of Computer Applications*, vol. 43, no. 17, pp. 12–16, 2012.

[11] D. Uppal, R. Sinha, V. Mehra, and V. Jain, "Malware detection and classification based on extraction of api sequences," in *2014 International conference on advances in computing, communications and informatics (ICACCI)*. IEEE, 2014, pp. 2337–2342.

[12] L. E. S. Jaramillo, "Malware detection and mitigation techniques: Lessons learned from mirai ddos attack," *Journal of Information Systems Engineering & Management*, vol. 3, no. 3, p. 19, 2018.

[13] D.-Y. Kao and S.-C. Hsiao, "The dynamic analysis of wannacry ransomware," in *2018 20th International conference on advanced communication technology (ICACT)*. IEEE, 2018, pp. 159–166.

[14] F. Leon, "Curs învățare automată."

[15] Cuckoo Foundation, "Cuckoo sandbox," https://cuckoosandbox.org/, Accesat 11.03.2023.

[16] Any.run, "Malware research with any.run," https://any.run/why-us/, 2023, Ultima accesare: 11.03.2023.

[17] Fortinet, "FortiSandbox," https://www.fortinet.com/content/dam/fortinet/assets/data-sheets/FortiSandbox.pdf, 2023, Ultima accesare: 11.03.2023.

[18] K. Baker, "Malware analysis." [Online]. Available: https://www.crowdstrike.com/cybersecurity-101/malware/malware-analysis/

[19] T. Alrawashdeh, R. Alshammari, and A. Almomani, "Combined dynamic multi-feature and rule-based behavior for accurate malware detection," *Journal of Cybersecurity Education, Research and Practice*, vol. 2020, no. 1, pp. 16–28, 2020.

[20] M. Alazab, R. Layton, S. Venkataraman, and P. Watters, "Malware detection based on structural and behavioural features of api calls," *International Cyber Resilience conference*, 2010.

[21] T. A. Abdurrahman Pektaş, "Malware classification based on API calls and behaviour analysis," *Information Security*, vol. 12, no. 2, pp. 107–117, 2018.

[22] M. Dredze, K. Crammer, and F. Pereira, "Confidence-weighted linear classification," in *Proceedings of the 25th international conference on Machine learning*, 2008, pp. 264–271.

[23] A. Cannarile, F. Carrera, S. Galantucci, A. Iannacone, and G. Pirlo, "A study on malware detection and classification using the analysis of api calls sequences through shallow learning and recurrent neural networks," 2022.

[24] Q. Q. Mingdong Tang, "Dynamic API call sequence visualisation for malware classification," *Information Security*, vol. 13, no. 4, pp. 367–377, 2019.

[25] K. Rieck, P. Trinius, C. Willems, and T. Holz, "Automatic analysis of malware behavior using machine learning," *Journal of computer security*, vol. 19, no. 4, pp. 639–668, 2011.