Hibridna metoda otkrivanja zlonamjernih programa

Gržinić, Toni

Sveučilište u Zagrebu

FAKULTET ORGANIZACIJE I INFORMATIKE VARAŽDIN

Toni Gržinić

Hibridna metoda otkrivanja zlonamjernih programa

DOKTORSKI RAD

Varaždin, 2017.

Sveučilište u Zagrebu

FAKULTET ORGANIZACIJE I INFORMATIKE VARAŽDIN

Toni Gržinić

Hibridna metoda otkrivanja zlonamjernih programa

DOKTORSKI RAD

Mentori:
prof.dr.sc. Mirko Čubrilo
doc.dr.sc. Tonimir Kišasondi

Varaždin, 2017.

University of Zagreb

FACULTY OF ORGANISATION
AND INFORMATICS VARAŽDIN

Toni Gržinić

Hybrid method of detecting malware

DOCTORAL THESIS

Supervisors:
prof.dr.sc. Mirko Čubrilo
doc.dr.sc. Tonimir Kišasondi

Varaždin, 2017

PODACI O DOKTORSKOM RADU

I. AUTOR

Ime i prezime	**TONI GRŽINIĆ**
Datum i mjesto rođenja	**19. prosinca 1985., Pula**
Naziv fakulteta i datum diplomiranja	**Fakultet organizacije i informatike Varaždin,**
Sadašnje zaposlenje	**Diverto d.o.o., Zagreb**

II. DOKTORSKI RAD

Naslov	**Hibridna metoda otkrivanja zlonamjernih programa**
Broj stranica, slika, tabela, priloga, bibliografskih podataka	**110 stranica, 16 slika, 12 tablica, 3 priloga, 109 bibliografskih podataka**
Znanstveno područje i polje iz kojeg je postignut akademski stupanj	**Društvene znanosti, informacijske znanosti**
Mentor i voditelj rada	**Prof.dr.sc. Mirko Čubrilo** **Doc.dr.sc. Tonimir Kišasondi**
Fakultet na kojem je rad obranjen	**Fakultet organizacije i informatike**
Oznaka i redni broj rada	**138**

III. OCJENA I OBRANA

Datum sjednice Fakultetskog vijeća na kojoj je prihvaćena tema	**24.1.2017.**
Datum predaje rada	**28.8.2017.**
Datum sjednice Fakultetskog vijeća na kojoj je prihvaćena pozitivna ocjena rada	**21.11.2017.**
Sastav Povjerenstva koje je rad ocijenilo	**Prof.dr.sc Marin Golub** **Prof.dr.sc. Željko Hutinski** **Prof.dr.sc. Mirko Maleković**
Datum obrane	**13.12.2017.**
Sastav Povjerenstva pred kojim je rad obranjen	**Prof.dr.sc Marin Golub** **Prof.dr.sc. Željko Hutinski** **Prof.dr.sc. Mirko Maleković**
Datum promocije	

Predgovor

Dragi čitatelju ispred tebe stoji doktorska disertacija u koju je uloženo puno kave i čajeva, dugih razgovora, energije, kodiranja te je pisana u sitne sate, vikende i blagdane. Put nije uvijek bio jednostavan te je bilo trenutaka kada sam mislio da nikada neće doći vrijeme pisanja ovog predgovora, čak i kada je konačni cilj bio blizu. Na svu sreću kroz ovaj put pratili su me mnogi prijatelji i obitelj koji su ovaj dugogodišnji proces učinili lakšim i bez kojih ove disertacije vjerojatno nikada ne bi bilo.

Prvo bih se zahvalio supruzi Ivani i kćeri Viti koje su dijelile sa mnom lijepe i ružne trenutke. Drage moje cure, nadam se da ću nadoknaditi vaše strpljene i vrijeme. Hvala vam na podršci!!

Zahvala ide i mojim roditeljima Davoru i Arieti te bratu Ninu, koji nikada nisu prestali vjerovati u mene. Također, nonićima Maliju i Toninu te nonama Dori i Lauri zahvaljujem na podršci i poticaju tijekom mojeg školovanja. Hvala na razumijevanju i strpljenu roditeljima moje supruge, Sulejmanu i Marijani.

Zahvalio bih se kolegama iz Diverta, a posebice direktoru Boži Šariću.

Bez žustrih rasprava s kolegom dr.sc. Markom Velićom moj doktorat sigurno bi trajao duže. Također, Marko, hvala ti na uvođenju u svijet znanstvenog publiciranja.

Zahvalio bih se mentorima prof.dr.sc. Mirku Čubrilu i doc.dr.sc. Tonimiru Kišasondiju koji su mi pomogli svojim savjetima i iskustvom. Posebno bih zahvalio mentoru Kišasondiju koji mi je ustupio na korištenje infrastrukturu Laboratorija za otvorene sustave i sigurnost za provođenje eksperimenata.

Posljednja, ali ne i najmanje bitna zahvala ide svima koje sam izostavio, a bar jednom su slušali moj tijek misli vezan za ovu disertaciju :).

U Zagrebu, 4.12.2017.

Sažetak

Maliciozni ili zlonamjerni programi predstavljaju danas najveću prijetnju poslovnim organizacijama diljem svijeta, a razvijaju se prvenstveno s ciljem krađe podataka te omogućuju krađu podataka i novaca od svojih žrtava. U svijetu gotovo da ne postoji poslovna organizacija ili kućni korisnik koji se nije susreo s malicioznim kodom, a u novije vrijeme brojnim štetama prouzročenim zlonamjernim programima pribrajaju se i slučajevi iz Hrvatske. Zadnjih godina zlonamjerni programi poput *ransomwarea* kriptiraju podatke na računalima tražeći otkupninu za vraćanje podataka ili bankarskih trojanaca koji ubacujući se u dobronamjerne programe presreću korisničku komunikaciju s e-bankarstvom te kradu sredstva s bankovnih računa. Za razliku od ciljanih napada kod kojih je potrebno veliko ulaganje zbog otkrivanja nepoznatnih ranjivosti i planiranja provedbe samog napada, logika uobičajenih zlonamjernih programa je vrlo jednostavna - razvijeni su da mogu učinkovito zaraziti što veći broj dostupnih računala. Najpoznatija zaštita od zlonamjernih sadržaja u posljednjih dvadesetak godina su antivirusni alati. Njihov detekcijski mehanizam temelji se na potpisima (engl. *signatures*) i heuristikama. Analitičari zaposleni u sigurnosnim tvrtkama, analizom zlonamjernih programa dolaze do potpisa, odnosno obrazaca koji prepoznaju zlonamjerne programe. Slično je i s heurističkim metodama, koje predstavljaju pravila kojima se otkrivaju općenite varijante zlonamjernih programa. Problem kod trenutnih metoda detekcije je što uvelike ovise radu i pronicljivosti analitičara koji analiziraju prethodno nepoznate varijacije zlonamjernih programa. Također, razvojem i povećanjem broja varijacija različitih zlonamjernih programa povećavaju se i baze potpisa, koje zbog svog velikog rasta predstavljaju neodrživi model za prepoznavanje novih zlonamjernih programa.

Cilj ovog istraživanja je izgradnja nove metode koja omogućuje prepoznavanje zlonamjernih programa bez korištenja potpisa. Nova metoda opisana u ovom istraživanju koristi statičku i dinamičku analizu programa, tj. dvije metode koje se međusobno nadopunjuju te daju različite poglede na funkcionalnosti i namjeru analiziranog programa.

Nova, hibridna metoda, koristi tri osnovna klasifikatora C_S, C_{D1} i C_{D2} za konačnu klasifikaciju prirode programa. Osnovni klasifikatori naučeni su na stratificiranom uzorku zlonamjernih programa TRAIN-1 koji se pojavio u razdoblju od 2011. - 2016. godine te na popularnim dobronamjernim programima. Cjeloviti skup programa iz kojeg je izuzet stratificirani uzorak prikupljen je s otvorenih izvora na internetu te su ručno dodani zlonamjerni programi koji se se koristili u APT (engl. *advanced persistent threat*) napadima. Stratificirani skup TRAIN-1 sadrži 2064 zlonamjernih programa te 980 benignih programa. Kako bi se testirala učinkovitost razvijenog hibridnog klasifikatora prikupljen je i neovisni skup TEST-1 koji sadržava zlonamjerne programe koji su se pojavili tijekom provođenja istraživanja, a koje popularni antivirusni alati nisu bili u mogućnosti detektirati.Prilikom izbora adekvatnih metoda strojnog učenja za osnovne klasifikatore evaluirane su sljedeće metode strojnog učenja: logistička regresija, jednostavni Bayesov klasifikator, metoda potpornih vektora, stabla odlučivanja i metoda nasumične šume. Metode strojnog učenja uspoređene su s ciljem utvrđivanja standardnog algoritma koja postiže najveću točnost klasifikacije na danom skupu podataka. Klasifikator C_S koristi značajke dobivene statičkom analizom *Portable Executable* datoteke, klasifikator C_{D1} koristi značajke vezane za karakteristike poziva prema operacijskom sustavu dok klasifikator C_{D2} koristi podatke o redoslijedu poziva prema operacijskom sustavu. Prilikom izrade hibridne metode slijedile su se dobre prakse u dubinskoj analizi podataka pri čemu su transformirane značajke osnovnih klasifikatora te je uspoređena točnost metoda strojnog učenja korištenjem cjelovitih skupova značajki s reduciranim skupovima značajki dobivenim metodama rekurzivne eliminacije (RFE) i glavnih komponenata (PCA). Hibridni klasifikator C_H koristi se za konačnu klasifikaciju vjerojatnosti pripadnosti pojedinoj klasi, tj. vjerojatnosti da je program zlonamjeran ili dobronamjeran, za svaki od osnovnih klasifikatora. Ovakvo slaganje klasifikatora omogućuje proširivanje same hibridne metode novim klasifikatorima koji se mogu koristiti za specifične namjene.

Za razliku od sličnih istraživanja hibridna metoda ne kombinira značajke statičke i dinamičke analize već zasebne klasifikatore svake pojedine analize. Korištenjem metode slaganja klasifikatora u hibridnoj metodi omogućuju se bolji rezultati zbog korištenja međusobno različitih značajki u osnovnim klasifikatorima te bolje prepoznavanje novih i prethodno nepoznatih varijacija zlonamjernih programa od trenutačno korištenih i referentnih metoda. Razvijeni hibridni klasifikator postiže bolje rezultate od pojedinačnih, osnovnih, klasifikatora C_S, C_{D1} i C_{D2} zbog korištenja različitih značajki i samih klasifikatora što u konačnici rezultira točnijom odlukom. Sama pretpostavka veće točnosti metode ansambla odnosno izbora gomile u odnosu pojedinačne odluke poznata je još od Galtonovog esejea *Vox populi* [30] i sama pretpostavka je provjerena u mnogim istraživanjima, ipak uspješnost samih ansambla u odnosu na pojedinčne klasifikatore uvelike varira te u domeni detekcije zlonamjernih programa nije dovoljno istražena. Hibridna metoda je na

neovisnom skupu postigla veću točnost od antivirusnih alata, 98% naspram 86%, te bolje rezultate od sličnih referentnih istraživanja.

Također, skupovi značajka koji se koriste u osnovnim klasifikatorima razlikuju se od sličnih istraživanja te su dodatno prošireni novim značajkama. Primjerice kod statičkih značajki S, provjerava se integritet sadržanih adresa, tj. postoje li nelogičnosti prilikom poravnanja izvršne datoteke u memoriji te sumnjive funkcije koje zlonamjerni programi koriste za tehnike skrivanja prilikom njihove analize. Dinamičke značajke D_1 opisuju kategorije poziva prema operacijskom sustavu poput: omjera uspješnosti poziva, njihovo vremensko trajanje, statistike korištenih argumentima u pozivu i sl.

Društvena opravdanost korištenja hibridne metode očituje se u obradi veće količine sumnjivih programa i sadržaja koje omogućuje rasterećenje rada analitičara te rano prepoznavanje opasnih programa koje mogu uzročiti znatne štete u energetskim postrojenjima, bankarskoj industriji i sličnim sustavima posebne namjene.

Ključne riječi: detekcija zlonamjernih programa, hibrida metoda, statička analiza programa, dinamička analiza programa, strojno učenje

Summary

Today malware is certainly the biggest threat to information security and business continuity of organizations around the world. The main reason why criminals develop malware is the ability to steal victims' money, sensitive data, or just to cause damages to their victims. Probably worldwide there is no company or home user that has not been infected by some variant of malware. Lately many organizations have had significant financial losses due to infections with malicious programs where Croatian companies were not an exception. Most popular malware families include ransomware, which encrypts victims' data demanding a ransom payment for recovering it, and banking trojans which intercept sensitive data in order to steal funds from bank accounts. Unlike advanced persistent threats, which are expensive both because they need zero day exploits and have to be carefully planned to efficiently infect their targets, generic malware is developed to infect commonly used operating systems. In the last twenty years computer systems are primarily protected against malicious content with the help of antivirus software. Antivirus software relies on the usage of signatures and heuristics to detect malware. While analyzing new malware samples malware analysts detect specific patterns also known as signatures, which are included in the signature database to enable detection of the specific malware by the antivirus. Similarly, heuristics represent a set of generic rules used to detect more efficiently various variations of similar malware. As we can see the process of detecting malware is costly because it heavily depends on human work done by malware analysts. Due to the mentioned cost of malware analysis and a exponential growth of new malware samples in the last years, signature databases slowly become an unsustainable model for detecting new malware.

The main goal of this research is to develop a method that is able to detect malware, in the common format Portable Executable used on Microsoft Windows, and without using signatures. The proposed hybrid method complementary uses results from static and dynamic analysis, which give different insights into functionalities and behavior of the analyzed program. Cuckoo Sandbox an open source malware analysis sandbox is used dynamic analysis of collected executables. The proposed hybrid method uses three basic classifiers C_S, C_{D1} and C_{D2} to classify programs, classifiers are individually trained on dataset TRAIN-1 that consists of malware samples from 2011 – 2016 stratified by their malware family and known benign programs such as executables from current Microsoft Windows versions and utility applications. The initial dataset, that included 19 877 malici-

ous programs, in the stratification process was reduced to dataset TRAIN-1 that included 2064 malicious programs and 980 benign programs. Various machine learning methods were compared and the method that yielded best accuracy on a given dataset was used as the basic classifier.

The following machine learning methods were evaluated for choosing the best basic classifiers C_S, C_{D1} and C_{D2}: Logistic Regression, Naïve Bayes, Decision Tree C 4.5. Support Vector Machine and Random Forest. Classifier C_S uses features extracted from the Portable Executable format, classifier C_{D1} uses features about specifics of systems calls intercepted during dynamic analysis, and C_{D2} uses the system calls sequence. Best practices were followed during the development of the hybrid method, for example data transformations and feature selection procedures were performed to identify appropriate feature subsets. Principal Component Analysis and Recursive Feature Elimination were used for dimensionality reduction and feature selection. The final hybrid classifier C_H uses class probabilities of basic classifiers for classification, i.e. probabilities that a program is malware or benign of each basic classifier. The used method of stacking classifiers or stacking generalization enables extension of the hybrid method with new classifiers designed for special purposes, for example introducing new basic classifiers for detecting malware that targets specific banking systems or industry controlled systems.

The presented hybrid classifier achieves better results than each individual basic classifier C_S, C_{D1} or C_{D2}. An independent dataset TEST-1 consisting of previously unseen malware was collected to test effectiveness of the hybrid classifier in real world scenarios. On the dataset TEST-1 the hybrid method achieved better accuracy results than state of the art antivirus tools, 98 % compared to 86 % (Kaspersky AV), and also achieved better results than similar benchmark research.

Combining individual basic classifiers using stacking generalization presents a novel contribution of this research, if we compare it with similar previous research that primarily combines static and dynamic features and not classifiers. Also, features used in basic classifiers differ from similar research and are extended with novel features. For example static analysis features S include features for checking addresses integrity, the related procedure checks the presence of wrong addressing when a program is loaded in memory, as well as presence of suspicious system calls that are commonly used for thwarting the malware analysis process (e.g. anti-debug, anti-vm, packing and others). Dynamic analysis features D_1 consist of system calls categories specifics like: successful system calls ratio, their duration, system calls arguments statistics and similar.

Using stacking generalization for combining classifiers in the hybrid method enables yielding better accuracy because basic classifiers use diverse features for building the final model, also this confirms the hypothesis that a combination of basic classifiers produces better results than a single classifier, and because of that the proposed hybrid classifier can detect new malware variants more efficiently than currently used methods.

Keywords: malware detection, hybrid method, static analysis of executables, dynamic analysis of executables, machine learning

Sadržaj

Popis slika

Popis tablica

Poglavlje 1

Uvod

U današnje vrijeme Internet osim svojih prednosti donosi i prijetnje. Od velikog broja mogućih tehnika napada najčešći su zlonamjerni programi. Tržište malicioznih sadržaja odvija se u "podzemlju", posredstvom zatvorenih kanala komunikacije i prvenstveno je ograničeno na profesionalne kriminalce koji često unajmljuju talentirane programere. Programeri, odnosno autori zlonamjernih programa, osim zlonamjernih radnji koje su svrha takvih programa, u njih ugrađuju napredne metode i mehanizme s ciljem zaobilaženja detekcije sigurnosnim alatima i uređajima. U novije vrijeme ovakvim visoko tehnološkim kriminalom bave se profesionalno organizirane kriminalne skupine. Glavna motivacija za razvoj zlonamjernih programa su: financijska korist, krađa povjerljivih informacija te u novije vrijeme korištenje Interneta u svrhu ratovanja (engl. *cyber warfare*).

Primjer korištenja zlonamjernih programa u svrhu internetskog ratovanja je sabotaža nuklearnog postrojenja u Iranu računalnim crvom *Stuxnet* [55], čija svrha je bila sabotaža iranskih nuklearnih postrojenja i usporavanje iranskog nuklearnog programa. Ciljanim napadom uništena je petina iranskih nuklearnih centrifugalnih jedinica i zaraženo je preko 60 000 računala. Do danas nitko nije preuzeo službenu odgovornost za napad, ali postoje sumnje da je napad organiziran od strane Sjedinjenih Američkih Država i Izraela. Glavni cilj napada bio je usporavanje iranskog nuklearnog programa i sprječavanje same ratne intervencije. Eksperti tvrde da je tehnološki napad na postrojenja bio djelotvorniji nego, primjerice, bombardiranje industrijskih postrojenja [57]. U današnje vrijeme takva vrsta, tzv. hibridnog, odnosno asimetričnog ratovanja postaje sve učestalija pojava te budućnost nam sigurno donosi učestlije premještanje rata u virtualne prostore interneta [8].

U zadnjih nekoliko godina poznate su prijetnje koje ciljaju korisnike internetskog ili e-bankarstva. Jedna od takvih prijetnja je bankarski trojanski konj Zeus kojime su zaraženi korisnici e-bankarstva diljem svijeta, ali i u Hrvatskoj [47]. Nastanjuje se na računalo žrtve te krade podatke vezane za e-bankarstvo, na način da mijenja izgled početne stranice e-bankarstva. Žrtve zaražene Zeusom čine mrežu zombi računala (botova) kojima upravlja centralni čvor (*botmaster*). U koordiniranoj akciji FBI-ja i Europola botnet Gameover

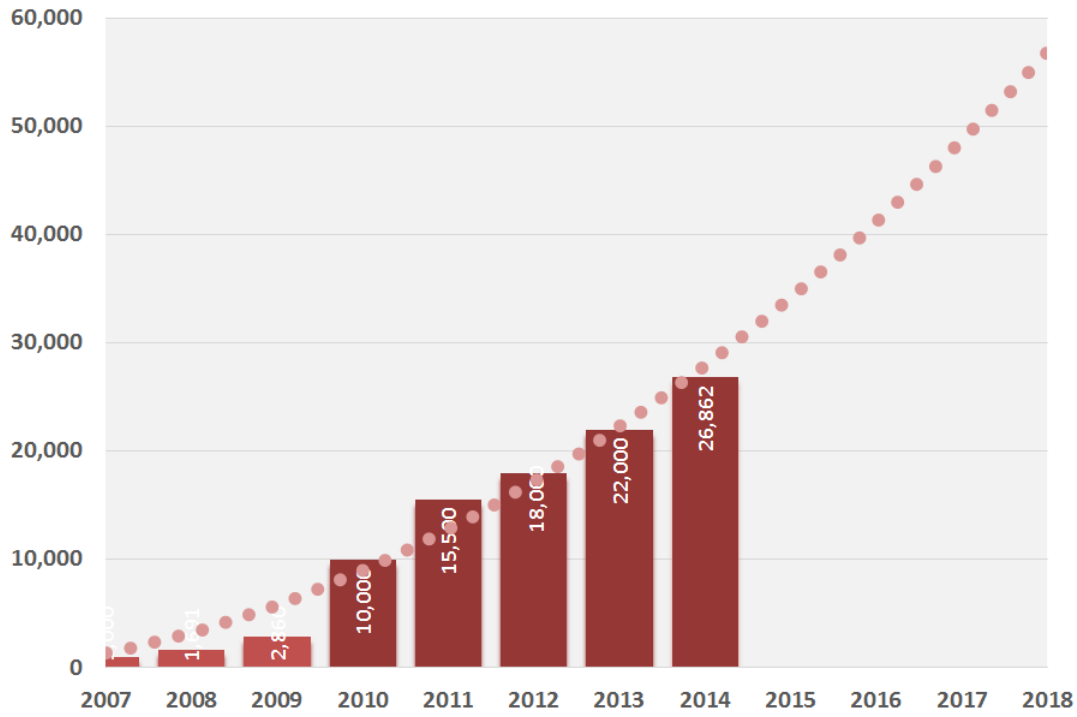
Zeus uspješno je ugašen u ljeto 2014., a nakon njegovog gašenja stručnjaci su procijenili da je s njim ukradeno više od 100 milijuna američkih dolara.

Zlonamjerni programi[1], poput Stuxneta i Zeusa, vrlo su složeni te je za njihovu analizu potrebna velika količina vremena i ljudskog rada. Zlonamjerne programe najčešće analiziraju antivirusne kompanije, odnosno njihovi analitičari (engl. *reverse engineers*) koji pokušavaju odgonetnuti što takav program radi i koja mu je namjena. Kako bi otežali posao analitičarima, tvorci zlonamjernih programa najčešće koriste različite tehnike kojima otežavaju analiziranje izvršne datoteke, odnosno programa. Primjerice, prikrivanjem kôda (engl. *obfuscation*) ili korištenjem različitih tehnika pakiranja, napadači dodatno otežavaju analizu ili je čine gotovo nemogućom. Analitičari nakon provedene analize nastoje u strojnom kôdu pronaći sekvencu koja jedinstveno označuje taj primjer zlonamjernog programa. Takav uzorak se u antivirusnim alatima naziva potpis (engl. *signature*). Antivirusni potpisi služe za prepoznavanje zlonamjernih programa [94]. Ukoliko antivirusni alat nema potpis za nepoznatu varijaciju zlonamjernog programa antivirusni alat neće prepoznati zlonamjerni program. No potpisi nisu jedina metoda kojom antivirusni alati prepoznaju zlonamjerne programe, već kao nadopunu njima antivirusni alati koriste heurističke metode [25]. Takve metode proizašle su iz iskustva analitičara i najčešće se implementiraju u obliku Ako-Onda pravila. Heuristike mogu prepoznati nepoznate varijacije zlonamjernih programa, ali njihov glavni nedostatak je veliki broj lažnih detekcija zbog kojih ih proizvođači koriste s oprezom i temeljito testiraju. Čest je slučaj da antivirusni programi klasificiraju malicioznim dobronamjerne programe (problem lažnih pozitiva), što predstavlja veliki problem u detekciji jer zahtjeva dodatni rad analitičara i ometa rad korisnika.

Prema podacima dostupnim u izvješćima tvrtke Symantec [100] i sličnim izvješćima o prijetnjama na Internetu, očito je broj potpisa eksponencijalno raste. Primjerice, prosječna baza potpisa antivirusnog alata iz 2014. godine sadržavala je 20 milijuna potpisa, dok prema procjeni prosječna baza antivirusnih potpisa u 2017. sadrži 50 milijuna potpisa za prepoznavanje zlonamjernih programa. Rast broja potpisa vidljiv je na slici 1.1, a podaci prikazuju neprestan porast u broju malicioznih sadržaja koje sigurnosni alati trebaju prepoznati.

Harely [25] tvrdi kako antivirusne tvrtke i njihovi alati nisu krivi za loše prepoznavanje zlonamjernih programa, već kako zlonamjerni programi postaju sve napredniji te je teško pratiti njihov razvoj. Čak i tvorac popularnog Norton Antivirusa, Brian Dye, naglašava kako antivirusni alati više nisu dovoljni za prepoznavanje malcioznih sadržaja [68].

[1] Kao sinonimi se još koriste *malver*, *zlonamjerni kôd*, *maliciozna izvršna datoteka* ili *maliciozni program*



Slika 1.1: Procjena broja potpisa u antivirusnim alatima, temeljeno na Symantecovim godišnjim izvještajima (Napomena: y-os je u 1000)

Očito je da napretkom malicioznih sadržaja i njihovom količinom, korištenje baza potpisa i njihovo održavanje postaje neodrživi model. Kao nadogradnja ili zamjena samih baza potpisa potrebno je razviti nove metode koje omogućavaju otkrivanje nepoznatih varijacije zlonamjernih programa na osobnim računalima. U znanstvenoj zajednici postoje napori da se razviju nove metode koje će pomoći u prepoznavanju zlonamjernih programa, a temelje se na statističkim metoda, metodama strojnog učenja i raspoznavanju uzoraka [106] [76].

1.1 Osvrt na terminologiju

Zbog specifičnosti samog područja i malog broja radova na hrvatskom jeziku u domeni prepoznavanja zlonamjernih programa ovdje su opisani termini korišteni u ovom radu. Istraživanje se bavi prepoznavanjem ili detekcijom malicioznih programa (od engl. *malicious*), kao sinonimi malicioznom programu mogu se koristiti termin zlonamjerni program ili malver (od engl. *malware*). Izbjegava se korištenje termina zloćudni program pošto je ćud svojstvo živih bića. Suprotno od zlonamjernih ili malicioznih programa, tj. programa namijenih nanošenju štete svojim žrtvama, su dobronamjerni ili benigni programi.

Prilikom korištenja metoda strojnog učenja, koje mogu nositi i naziv algoritmi strojnog učenja, spominju se značajke koje opisuju podatke. Značajke predstavljaju stupce u ma-

tricama prikupljenih podatka te se u literaturi nazivaju i atributima (engl. *attributes*). Iz prikupljenog skupa značajki možemo izdvojiti značajke za jedan ili više programa, izdvojene programe možemo zvati još instancama ili primjerima. Drugim riječima značajkama se opisuju primjeri koji se nalaze u skupu podataka. Prilikom korištenja nadziranih (engl. *supervised*) metoda strojnog učenja prvi korak je treniranje ili učenje koje se izvodi s ciljem podešavanja parametara korištene metode kako bi metoda uspješno generalizirala primjere prilikom procesa klasifikacije.

1.2 Motivacija

Motivacija za predloženo istraživanje proizašla je iz trenutnog dostignuća znanosti u području prepoznavanja zlonamjernih programa, ali i iz problema prepoznavanja malicioznog sadržaja u industrijskog praksi. Napadači redovito mijenjanju i smišljaju nove mehanizme zaobilaženja sigurnosnih alata. Sigurnosni alati često ne prepoznaju nove uzorke i varijacije postojećih zlonamjernih programa dok analitičari ne naprave analizu sumnjivog programa i ažuriraju bazu potpisa svojih alata. Još jedan problem kod korištenja potpisa u antivirusnim alatima je taj što veličina baza potpisa raste s rastom broja zlonamjernih programa. Budućnost donosi povećanje broja novih napada te samih baza potpisa, kao što je prethodno i navedeno, stoga oslanjanje isključivo na baze potpisa postaje neodrživi model za prepoznavanje novih malicioznih sadržaja.

Tvrtka LastLine [101] navodi kako, prema njihovim istraživanjima, jednostavnije varijante zlonamjernih programa trećina testiranih antivirusnih alata ne može prepoznati do dva mjeseca nakon njihovog pojavljivanja. U istom istraživanju navode da pri pojavi prethodno nepoznatog zlonamjernog programa, antivirusnim tvrtkama je u prosjeku potrebno od nekoliko sati do dva dana za njegovu uspješnu detekciju.

Istraživanja u trenutnom stanju znanosti koriste različite metode i pristupe kako bi riješili problem prepoznavanja novih zlonamjernih programa. Rijetka istraživanja [7] [82] [40] kombiniraju različite metode analize programa, a u literaturi se mogu susresti pojmovi poput hibridnih modela i hibridnih značajki koji predstavljaju modele i značajke nastale kombiniranjem različitih metoda analize programa.

Ovo istraživanje predlaže novu metodu kombiniranja klasifikatora koji koriste značajke dobivene različitim metodama analize programa, poput statičke i dinamičke analize, a njima se nastoje umanjiti nedostatci koji se pojavljuju kada se svaka metoda analize koristi pojedinačno.

1.3 Opis problema

Ovo istraživanje prvenstveno rješava problem prepoznavanja ili detekcije zlonamjernih programa. Detekcija zlonamjernih programa je klasifikacijski problem, u kojem svaki program može pripadati klasi zlonamjeran (maliciozan) ili dobronamjeran (benigan). Samo istraživanje bavi se rješavanjem ovog problema te izgradnjom okvira koji služi za analizu programa u formatu *Portable Executable*, koji je svojstven za operacijski sustav Microsoft Windows. Izgrađeni okvir za analizu zlonamjernih programa može se proširiti i na druge formate zapisa (skriptni jezici) te ostale izvršne programe različitih operacijskog sustava (GNU Linux, Android i sl.). Ipak, istraživanje je orijentirao isključivo na detekciju zlonamjernih programa za operacijski sustav Windows iz razloga što većina zlonamjernih programa [36] ciljaju upravo taj operacijski sustav zbog svoje rasprostranjenosti i primjenjivosti.

Danas klasični pristupi prepoznavanja zlonamjernih programa teško mogu prepoznati nove varijacije zlonamjernih programa, iz razloga što postoji velik broj različitih varijacija zlonamjernih programa koji za izbjegavanje sigurnosnih alata koriste razne metode sakrivanja, formate i vrste pakiranja. Pod klasičnim pristupima podrazumijevamo prvenstveno klasične antivirusne i anti-malver alate koji se oslanjaju na statičke potpise i heuristike, tj. pravila ponašanja. U novije vrijeme novi proizvodi na tržištu štite organizacije, ne samo koristeći statičke značajke, već posjeduju i zaštićene okoline (engl. *sandbox*). Pomoću zaštićenih okolina programi se analiziraju dinamički, tj. prikupljene dinamičke značajke prikazuju interakciju analiziranog programa s operacijskim sustavom. Logični slijed korištenih pristupa je postupno povezivanje ova dva tipa dobivenih značajki, tj. vrste analize programa, te stvaranje klasifikacijskog modela koji omogućuje prepoznavanje prirode programa.

U ovom istraživanju razvija se nova metoda koja koristi značajke dobivene statičkom analizom, ali i dinamičke značajke koje su dobivene prilikom izvršavanja programa u zaštićenoj okolini. Svaki skup značajki koristi se za izgradnju osnovnog klasifikatora (0-reda). Osnovni klasifikatori daju svoje predikcije prirode programa i koriste se u konačnom hibridnom klasifikatoru (1-reda). U literaturi je ova metoda poznata pod nazivom slaganje klasifikatora (engl. *stacking generalization*), koja u trenutačnom stanju znanosti nije primijenjena u domeni detekcije malcioznih programa. Slaganje klasifikatora omogućava veću fleksibilnost i proširivost od postojećih i usporedivih metoda [7] jer je moguće dodati nove klasifikatore specijalnih namjena i time poboljšati klasifikacijske mogućnosti za granične slučajeve.

Za razliku od dosadašnjih istraživanja hibridna metoda koristi prošireni skup statičkih značajki koje nisu samo direktno dobivene iz formata *Portable Executable*, u kojem se nalaze analizirani programi, već provjerava integritet zapisanih podataka sa zatečenim stanjem - tako se primjerice provjerava nalazi li se adresa ulazne točke programa (engl. *entrypoint*) zaista u izvršnoj sekciji *.text*. Uvezeni pozivi prema operacijskom sustavu označavaju se prema poznatim metodama za izbjegavanje detekcije i analize svojstvenim za zlonamjerne programe te se prepoznate metode koriste kao značajke u svrhu daljnje klasifikacije. U sličnim istraživanjima ne koriste se sumnjivi pozivi u svrhu klasifikacije ili prepoznavanja metoda izbjegavanja analize i detekcije.

Novina u odnosu na slična istraživanja kod dinamičkih značajki je korištenje podataka vezanih za argumente poziva te uzimanje u obzir njihovih statistika, vremenskog trajanja poziva i vremenskog razmaka između poziva. Također, osim samih poziva koriste se i njihove kategorije (npr. mrežni pozivi) koje daju općenitiju sliku o samom izvršavanju programa.

1.4 Hipoteze i ciljevi istraživanja

Cilj istraživanja je razviti novu metodu za prepoznavanje zlonamjernih programa koja koristi značajke statičke i dinamičke analize za klasifikaciju prirode programa. Predložena metoda koristi sustav više nadziranih klasifikatora, uz sljedeća obilježja:

- S obzirom na trenutno stanje u znanosti nova metoda mora biti bolja od trenutačno referentnih znanstvenih istraživanja: Andersona i suradnika [7] [6], Santosa i suradnika [82] te Islama i suradnika [40], što znači da nova metoda mora imati općenitu točnost veću od 90% i stopu lažnih pozitiva manju od 10% za prethodno nepoznate zlonamjerne programe.
- Nova metoda mora biti neovisna o potpisima antivirusnih alata, ali je potpise moguće implementirati u samu metodu.
- Nova metoda ne smije otpakiravati ili disasemblirati programe. Operacija otpakiranja je vremenski zahtjeva za analitičara te ju je vrlo teško generički implementirati zbog postojanja velikog broja slučajeva i vrsta pakiranja, dok je disasembliranje dobronamjernih programa krši uvjete korištenja takvih programa.
- Nova metoda temelji se na kombiniranju rezultata dobivenih iz različitih načina analize programa.
- Nova metoda može se koristiti u postojećim hodogramima analize programa u cilju otkrivanja prethodno nepoznatih zlonamjernih programa.

Hipoteze predloženog istraživanja su sljedeće:

1. Nova metoda ima općenitu točnost klasifikacije bez prethodnog otpakiravanja ili disasembliranja programskog koda, veću od 90% te stopu lažnih pozitiva manju od 10%.

2. Točnost klasifikacije zlonamjernih programa korištenjem hibridnog klasifikatora nadmašuje točnost pojedinačnog klasifikatora koji koristi značajke statičke ili dinamičke analize.

1.5 Pravni, etički i društveni aspekti predloženog istraživanja

Pravnih zapreka za korištenje prikupljenih uzoraka zlonamjernih programa nema, pošto su oni razvijeni od strane napadača te ne podliježu nikakvim ograničenjima osim njihove dostupnosti. Granični slučajevi su trojanski konji koji mogu biti umetnuti u komercijalne programe poput računalnih igara i popularnih uslužnih alata. Kako bi se izbjeglo korištenje takvih zlonamjernih programa prikupljeni skup je provjeren i takvi programi su iz njega izbačeni. Kod korištenja benignih programa, koji su dio operacijskog sustava Windows, poštovana su pravilima za korištenje za krajnje korisnike (*End user license agreement - EULA*). U pravilima korištenja stoji da je zabranjen bilo kakav oblik analize izvršne datoteke u svrhu stjecanja saznanja o internom načinu funkcioniranja samog programa. Privatnost korisnika nije narušena jer uzorci programa ne sadrže osobne podatke korisnika.

Kako bi se omogućilo ponavljanje eksperimenta i usporedbu rezultata, korišteni skupovi programa javno su objavljeni na Internetu [32]. Prilikom odabira zlonamjernih programa korištenih u istraživanju, izbjegnuto je pristrano biranje uzoraka, a iz cijelog skupa prikupljenih zlonamjernih programa izabran je stratificirani uzorak prema familiji kojoj zlonamjerni program pripada, što nije slučaj u sličnim istraživanjima.

Kako bi smanjili mogućnost pojave pretreniranosti (engl. *overfitting*) klasifikatora i rezultata koju mogu dovesti u zabludu čitatelja prilikom prikaza uspješnosti klasifikacije priložene su krivulje grešaka učenja tj. graf pristranosti i varijance (engl. *bias-variance plot*) za skup za učenje, koji se koristio u postupku unakrsne validacije.

Kao što je i prikazano na slici 1.1, količina zlonamjernih programa svake godine je u stalnom porastu te su mnoge poslovne organizacije diljem svijeta zbog loše mogućnosti prepoznavanja i neadekvatne zaštite pretrpjele znatne financijske štete. Iz tog razloga ovo istraživanje ima svoju društvenu opravdanost, koja se očituje u novoj unaprijeđenoj metodi, koja sigurnosnim analitičarima olakšava analizu nepoznatih programa i adekvatnu, tj.

pravovremenu reakciju na moguće ugroze. Također, razvijena metoda omogućuje prepoznavanje zlonamjernih programa bez korištenja potpisa koju je moguće ugraditi u postojeće sigurnosne uređaje i alate koji služe za prepoznavanje zlonamjernih programa u mrežnom okružju. Metodu je moguće koristiti za obradu velike količine zlonamjernih programa i prepoznavanje novih zlonamjernih programa koje je potrebno dodatno analizirati.

Poglavlje 2

Teorijski okvir

U teorijskom okviru definiran je pojam zlonamjernog programa i moguće vrste zlonamjernih programa koje se najčešće susreću u praksi. Također, opisan je standardni format izvršnih (engl. *Portable Executable*) datoteka u operacijskom sustavu Windows. Navedene su metode koje zlonamjerni programi koriste kako bi izbjegli detekciju sigurnosnim alatima. Opisane su vrste analiza izvršnih programa primijenjene u ovom istraživanju. Također, opisane su metode strojnog učenja korištene za klasifikaciju i pripadajuće mjere uspješnosti s kojima se procjenjuje kvaliteta klasifikatora. Dan je pregled metoda odabira značajki i redukcije dimenzionalnosti koje su korištene u ovom istraživanju.

2.1 Vrste zlonamjernih programa

Zlonamjerni program ili malver definira se kao program ili programska komponenta koja ostvaruje malicioznu namjeru njegovog tvorca, odnosno napadača [64] u cilju nanošenja štete ili krađe (podatka, novca i sl.). Važno je naglasiti da zlonnamjerni program nije propust u postojećom softveru, ali se propust može iskoristiti za širenje zlonamjernog programa. Zlonamjerni programi se razlikuju prema svojoj namjeri i svojstvima, poput metoda širenja ili prikrivanja. Na taj način možemo razlikovati sljedeće vrste zlonamjernih programa [27]:

1. **Virus** je zlonamjerni program koji se ugrađuje u druge programe i aktivira pri pokretanju. Za njegovo aktiviranje ili prenošenje potrebna je korisnikova akcija (za razliku od crva koji ima samostalan mehanizam širenja). Virusi mogu imati različite namjene: od nanošenja štete do prikaza reklama ili stvaranja botnet mreže.

2. **Crv** (engl. *worm*) je vrsta zlonamjernog program koji se širi kroz računalne mreže (lokalne i Internet) iskorištavajući postojeće ranjivosti te koristi mehanizam samoreplikacije kako bi samostalno zarazio ostala računala na mreži. Crvi često u tu svrhu koriste dodatni sadržaj (engl. *payload*) kojime se iskorištava ranjivost u programskim komponentama ili samom operacijskom sustavu.

3. **Trojanac** (engl. *trojan horse*) je vrsta zlonamjernog programa koja se pretvara da radi nešto korisno i ima karakteristike benignih programa. Često dolaze u obliku dodataka za web preglednike, čuvara zaslona ili računalnih igara sumnjivog porijekla te imaju svojstvo ubacivanja u dobronamjerne programe. Nakon instalacije maliciozni dio skida dodatni zlonamjerni program ili mijenja postavke operacijskog sustava.

4. **Bot** je vrsta zlonamjernog programa koja omogućava njegovom vlasniku, kojeg još nazivamo *botmaster*, kontrolu nad zaraženih sustavom.

5. ***Rootkit*** je vrsta zlonamjernog programa koja ima mogućnost sakrivanja svoje prisutnosti unutar operacijskog sustava. Za potrebe svog sakrivanja *rootkiti* koriste funkcije jezgre operacijskog sustava te iz tog razloga mogu biti implementirani u obliku upravljačkog programa (engl. *driver*).

6. ***Spyware*** je vrsta zlonamjernog programa koja nadzire žrtvinu aktivnost, tj. može nadzirati aktivnosti poput korisnikovih aktivnosti na tipkovnici (engl. *keylogging*), sakupljati osjetljive podatke poput lozinki ili pristupnih podataka internetskog bankarstva i sl. Spyware se može nastaniti u operacijskom sustavu iskorištavanjem neke ranjivosti ili može biti dio nekih legitimnih programa ili trojanca.

7. **Ransomware** je noviji oblik malicioznog programa. Ransomware kriptira datoteke na tvrdom disku i time onemogućuje normalno korištenje operacijskog sustava, nakon kriptiranja podatka od žrtve se traži otkupnina koju je potrebno platiti u kratkom vremenskom roku kako bi se podaci dekriptirali. Širi se na sličan način kao i crv, na računalo dolazi preuzimanjem malicioznih datoteka putem Interneta, e-porukama koje navode korisnika da otvori njihov privitak ili iskorištavanjem ranjivosti ranjive softverske komponente.

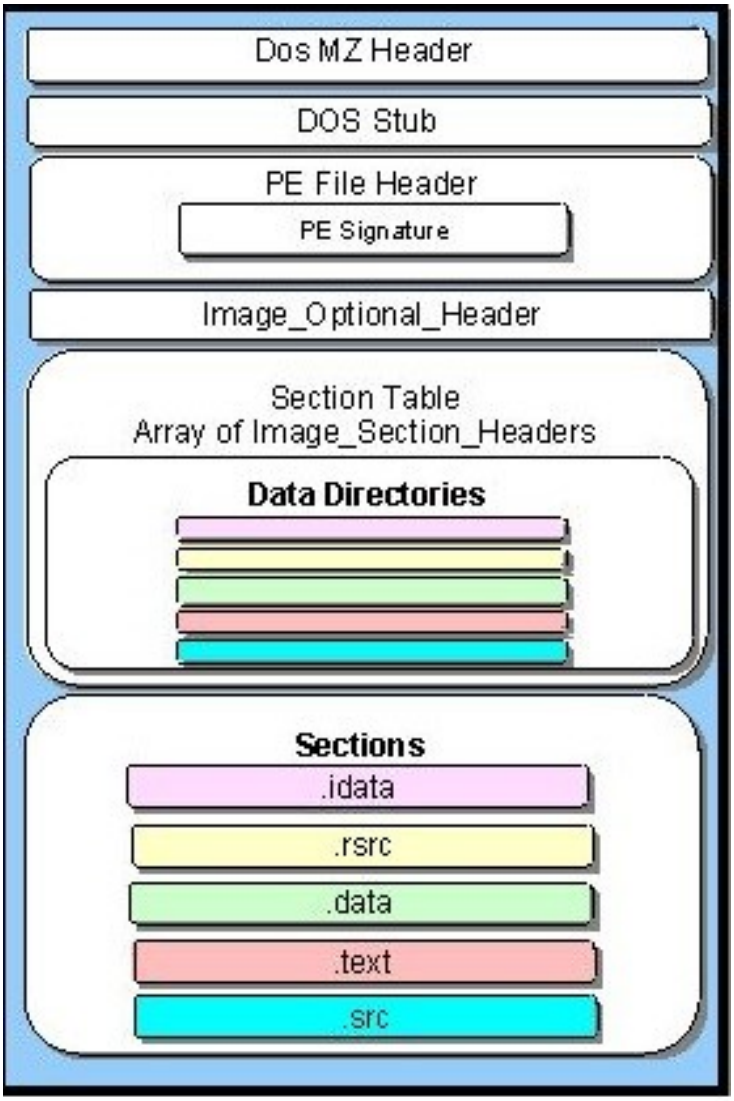
2.2 Portable Executable - standardni oblik izvršnih datoteka u operacijskom sustavu Windows

Najčešća meta napada su korisnici Windows operacijskog sustava, najpopularnijeg operacijskog sustava na osobnim računalima. Izvršne datoteke i bibliotečne funkcije (engl. *dynamic link library* ili DLL) u operacijskom sustavu Windows imaju specifičan format nazvan *Portable Executable* ili skraćeno PE [70]. PE je kompatibilan unazad, tj. ima mogućnost izvršavanja na različitim verzijama operacijskih sustava tvrtke Microsoft. Sastoji se od zaglavlja i sekcija te sadrži potrebne strukture podataka, pomoću kojih učitavatelj (engl. *loader*) operacijskog sustava učitava izvršni kôd u radnu memoriju računala. Učitana izvršna datoteka u radnoj memoriji računala i njen zapis na disku imaju istu strukturu. Struktura formata PE prikazna je na slici 2.1.

Svaka PE datoteka sadrži: DOS zaglavlje koje je sadržano zbog kompatibilnosti, te NT (*Portable Executable*) zaglavlje koje sadrži dvije osnovne strukture: *Image File Header* i *Image Optional Header* [61] . *Image File Header* je jednostavna struktura koja ima pohranjene osnovne informacije o izvršnoj datoteci, kao što su: platforma za koju je program namijenjen (npr. i386), broj sekcija, pokazivač na simbole, broj simbola, veličinu opcionalnog zaglavlja i dr. Nakon nje slijedi struktura *Image Optional Header* koja sadrži sve potrebne informacije o smještaju sekcija u memoriji, gdje se program počinje izvršavati, veličine stoga i hrpe i sl. Prije samih informacija o sekcijama nalazi se struktura s opisima sekcija (engl. *Image Section Header*), u kojoj se nalaze podaci poput duljine sekcija i adresa potrebnih za pristup sekcijama. Sekcije PE datoteke sadržavaju izvršni kôd, podatke i druge resurse (slike, znakovne nizove i sl.) potrebne za izvršavanje samog programa.

Uvezene biblioteke definirane su opisnom listom koja pokazuje na tablicu uvezenih funkcija (engl. *Import Address Table* ili IAT). Svaki zapis u njoj opisuje biblioteku koju promatrani program koristi. U njoj se nalaze dva ključna pokazivača (*OriginalFirstThunk* i *FirstThunk*) koji pokazuju na dvije tablice *Import Name Table* ili INT i *Import Address Table* ili IAT. Obje tablice se sastoje od *Image Thunk Data* struktura koje pokazuju na uvezene funkcije u listi *Image Import by Name*. Tablice IAT I INT pokazuju na ista polja

i naočigled imaju istu funkciju, no ipak prilikom učitavanja u memoriju u IAT tablicu se zapisuju stvarne memorijske adrese dok INT tablica sadrži relativne adrese prema funkcijama.



Slika 2.1: Struktura formata Portable Executable (preuzeto iz [38])

PE format ima mogućnost stvaranja biblioteke funkcija koju uvoze drugi programi. Za tu namjenu zadužena je, slično kao i kod IAT-a, tablica izvezenih funkcija (engl. *Export Address Table*) koja sadrži funkcije namijenjene korištenju u drugim programima.

Najvažnije sekcije u PE formatu su [90]:

1. U sekciji *.text* se nalaze instrukcije koje izvršava procesor, a koje sadržavaju izvršni kod programa. Ovo sekcija je važna jer sadržava izvršni kôd, dok sve druge sekcije sadrže resurse ili podatke. Sekcija *.text* može imati još i oznaku *code*.

2. Sekcija *.rdata* sadrži informacije o uvezenim i izvezenim funkcijama. Ova sekcija može sadržavati podatke namijenje samo čitanju. Sekcija *.rdata* može sadržavati sekcije *.idata* i *.edata* koje sadrže uvezene i izvezene funkcije.

3. Sekcija *.data* sadrži globalno dostupne podatke. Osim globalnih podatka sadržava i konstante koje se inicijaliziraju prilikom kompajliranja programa. Lokalne varijable ne spremaju se u sekcije već se za njih koristi memorijska struktura stoga, koja se nalazi u dretvi pokrenutog programa.

4. Sekcija *.rsrc* sadrži resurse; može sadržavati ikonu programa, slike koje se koriste u programu ili tekst.

2.3 Metode analize izvršnih datoteka

Analitičari najčešće analiziraju maliciozni sadržaj pomoću *statičke* ili *dinamičke* analize. Iako se ove dvije analize koriste odvojeno, one se često nadopunjavaju, u smislu da rezultat jedne analize može dati potrebne informacije za kvalitetno provođenje druge vrste analize.

Statička analiza podrazumijeva prebacivanje izvršne datoteke u jezik najniže razine, Asembler, te analizu kôda najniže razine bez njenog izvršavanja. Kao što smo i prije naveli, cilj analize zlonamjernog programa je prepoznati namjeru i funkcionalnosti zlonamjernog programa te identificirati jedinstveni potpis koji služi za detekciju zlonamjernog progama u npr. antivirusnom alatu. Statička analiza sastoji se najčešće od provjere strukture i sadržaja unutar izvršne datoteke. Sljedeći korak je pregled izvršne datoteke u nekom disasemblerskom alatu, primjerice IdaPro [35]. Disasemblerski alat prevodi kompajlirani kôd u asemblerske instrukcije. Same asemblerske instrukcije IdaPro prikazuje u obliku grafa tijeka podataka (engl. *control flow graph*), što je praktično za utvrđivanje mogućih grananja u programu. Kako autori zlonamjernih programa često koriste različite oblike prikrivanja, statička analiza nije učinkovita pa se sve više koristi dinamička analiza za otkirvanje funkcionalnosti zlonamjernih programa.

Dinamička analiza podrazumijeva analizu ponašanja programa, npr. pokretanjem programa u izoliranoj (zaštićenoj) okolini ili kontroliranjem kroz *debugger*. Za potrebe dinamičke analize analitičari često koriste *debugger* koji omogućuje kontroliranje tijeka izvršavanja programa. Najčešće korišteni *debuggeri* su OllyDbg [109] i WinDbg [103].

OllyDbg ima mogućnost pauziranja na izabranoj instrukciji (engl. *breakpoint*), primjerice instrukciji gdje počinje procedura za odkodiravanje prikrivenog sadržaja. Postupnim izvršavanjem instrukcija moguće je "otpetljati" enkripcijski postupak i vidjeti što sadržava kodirani (prikriveni) sadržaj. Na sličan način je moguće otpakirati zapakirani program. U tom procesu analitičaru je cilj pronaći rutinu raspakiravanja te prepoznati stvarni početak izvršavanja programa (engl. *original entry point*). Dinamičkom analizom mogu se nadzirati pozivi prema funkcijama operacijskog sustava, parametri koji se prosljeđuju funkcijama, tijek podataka prilikom izvršavanja programa i sl. Analitičari dinamičku analizu najčešće provode u virtualnoj i zaštićenoj okolini. Često se u tu svrhu koriste alati Sysinternals, kao što je ProcessMonitor [79] koji nadzire operacije nad *registryjem*, promjene nad datotekama i mrežnu aktivnost te Process Explorer [78] koji služi za napredno nadziranje aktivnih procesa.

Zaštićene okoline (engl. *sandbox*) su popularan oblik analize izvršnih datoteka koje nastoje automatizirati dinamičku analizu zlonamjernih programa. Cilj im je snimiti interakciju zlonamjernog progama s operacijskim sustavom, tj. pozive operacijskog sustava uz pomoć kojih je moguće otkriti funkcionalnosti i namjeru zlonamjernog programa te se ti snimljeni pozivi mogu koristiti za daljnju klasifikaciju prirode programa. Većina zaštićenih okolina koristi neki virtualizacijski alat ili emulator u kojemu se izvršava program i snima se njegova interakcija s operacijskim sustavom. Neke od popularnih i korištenih zaštićenih okolina jesu: Anubis [39], CWSandbox [102], JoeSandbox [44], Hybrid Analysis [86] i Cuckoo Sandbox [24], koji je ujedno najpopularniji okvir za provođenje dinamičke analize slobodnog kôda i koristi se u ovom istraživanju.

2.3.1 Značajke malicioznih programa

Kao što je prethodno navedeno, autori malicioznih programa koriste različite tehnike kako bi analitičarima otežali analizu. Isto tako, navedene tehnike služe kako bi maliciozni programi zaobišli detekciju antivirusnih alata [50]. Autori malicioznih programa koriste sljedeće metode prikrivanja prilikom njihove izrade [12]:

1. **Pakiranje** stvara od izvršnog programa sintaktički drugačiju instancu zadržavajući istu semantičku vrijednost. Drugim riječima, dva programa (pakirani i originalni) jednaka su po funkcionalnosti, ali su različiti po obliku. *Paker* je alat koji stvara semantički jednaku varijaciju programa. Svaki zapakirani program posjeduje rutinu raspakiravanja koja je zadužena da tijek izvršavanja dođe do malicioznog dijela kôda. Sam postupak otpakiravanja izvodi se prilikom pokretanja programa u memoriji računala. Pakiranje često može koristiti sažimanje izvršne datoteke. Popularni alati koji autori malicioznih programa koriste za pakiranje su: UPX, ASPack, VMProtect, WinUpack itd.

2. **Kriptiranje kôda** je postupak u kojem se dio izvršnog kôda kriptira, odnosno pomoću generiranih kriptografskog ključa mijenja se oblik program i time izbjegava detekcija sigurnosih alata . Takav maliciozni program koristi neki enkripcijski algoritam ili postupak kodiranja, te ukoliko je potrebno, enkripcijski ključ koji je sadržan u samom programu ili se generira pri njegovom pokretanju. Dio izvršnog kôda je kriptiran, odnosno analitičaru nečitljiv, do same faze dekripcije koja se događa pri izvršavanju malicioznog programa. Analitičar se u ovom scenariju koristi alatom koji omogućava kontrolu izvršavanja programa (engl. *debugger*) kako bi prošao fazu dekripcije i otkrio što je zapisano u kriptiranom sadržaju. Najjednostavniji primjer enkripcije je korištenje funkcije ekskluzivnog ILI (XOR) s nekom proizvoljnom vrijednošću koja predstavlja ključ. Prilikom izvršavanja malvera koristi se ključ, koji je zapisan u njemu, kako bi se dekriptirao maliciozni dio. Prilikom završetka izvođenja maliciozni program može kopirati samog sebe i generirati novi ključ za novu instancu svog programa nasljednika. Autori malicioznih programa mogu koristiti i druge metode kodiranja ili enkripcijske algoritme poput: Base64, ROT13, RC4, AES i slične.

3. **Polimorfni i metamorfni programi** mijenjaju svoj oblik prilikom svake svoje nove varijacije ili kopiranja. Polimorfni program mijenja svoj enkripcijski postupak pri svakoj novoj infekciji. Često se može mijenjati enkripcijski ključ koji se nalazi u samom izvršnom programu ili se generira pri izvođenju. Neki polimorfni programi mogu mijenjati vrstu algoritama za enkripciju uz promjenu samih ključeva. Ovakva vrsta malicioznog programa ima u sebi transformacijski postupak koji izabire enkripcijske algoritme i koji kriptira buduće varijacije malicioznog programa. Metamorfni programi su najsloženija vrsta malicioznih programa koja sama sebi mijenja izvorni kôd i postupke za otpakiravanje, a zadržavaju iste funkcionalnosti.

4. **Ostale tehnike** nastoje otežati ručnu analizu naprednim tehnikama koje sprečavaju disasembliranje, prepoznaju korištenje *debuggera* ili zaštićene okoline. Tehnike koje otežavaju proces disasembliranja (engl. *anti-disassembly*) mogu umetnuti nepotrebne instrukcije ili pravi kôd koji nema nikakvu funkcionalnost, preimenovati pojedine dijelove kôda, zapisivati vrijednosti u registre koje se kasnije nikada ne koriste, ili nepotrebnim skokovima otežavati praćenje tijeka programa. Različite *anti-debug* tehnike ometaju dinamičku analizu prepoznavanjem kontrole *debuggera* nad analiziranim malicioznim programom. Maliciozni program koji koristi *anti-debug* tehniku ili prepoznaje virtualnu okolinu može odgoditi svoje izvršavanje i ponašati se kao benigni program.

2.4 Klasifikacija malicioznih programa korištenjem metoda iz domene strojnog učenja

Klasifikacija zlonamjernih programa može se promatrati kao problem u domeni strojnog učenja. Prema Mitchellovoj definiciji [63], za algoritam se kaže da uči iz iskustva E u odnosu na zadaću T, s mjerom uspješnosti P, ukoliko se uspješnost za zadaću povećava s iskustvom. U kontekstu strojnog učenja prepoznavanje zlonamjernih programa iz skupa izvršnih programa predstavlja zadaću (engl. *task*). Kako bi klasificirali program trebamo svaki primjer iz skupa programa opisati, tj. opis sadrži značajke programa koje opisuju svojstva programa. Najjednostavniji primjer značajke za ovaj klasifikacijski problem je veličina programa u megabajtima.

Kako bi klasificirali uzorke programa potrebno je imati označen skup, odnosno svaki program treba imati dodijeljenu klasu kojoj pripada. U našem problemu svakom programu se dodijeljuje klasa - dobronamjerni (*benigni*) ili zlonamjerni (*maliciozni*). Postupak nadziranog učenja (engl. *supervised learning*) koristi označeni skup za podešavanje parametra korištene metode strojnog učenja, odnosno naučeni model ima za cilj što bolje generalizirati primjere u dane klase kako bi postigao što veću točnost. Model se još može nazvati i klasifikatorom. Skup svih programa možemo podijeliti u dva dijela: skup za učenje (engl. *training*), koji služi za podešavanje težina modela i skup za testiranje koji služi za procjenu uspješnosti testiranog algoritma strojnog učenja. Uspješnost algoritma može biti predstavljena mjerom općenite točnosti (engl. *accuracy*) koja predstavlja omjer programa, malicioznih i benignih, koji su dobro klasificirani.

Naučeno znanje iz podataka modeli mogu predstavljati u raznim oblicima. Primjerice, Flach [29] razlikuje:

- Geometrijske modele (npr. logistička regresija) koji znanje predstavljaju u obliku ravnine. Primjerice, logistička regresija odvaja klase pravcem.
- Vjerojatnosne modele (npr. jednostavni Bayesov klasifikator) koji znanje predstavljaju u obliku vjerojatnosti.
- Logičke modele (npr. stabla odlučivanja) koji znanje predstavljaju u obliku pravila ili stabla odlučivanja.
- Modele grupiranja (npr. k-najbližih susjeda ili hijerarhijsko grupiranje) koji koriste mjere udaljenosti kako bi grupirali instance iz skupa prema sličnosti. Karakteristični su pretežno za nenadzirano učenje.

U dostupnim istraživanjima za problem klasifikacije zlonamjernih programa korišteni su razni algoritmi strojnog učenja. Najčešće korišteni algoritmi su: metoda potpornih vektora SVM [76] [51], asocijacijska pravila [106], skriveni Markovljevi modeli [7], Bayesove mreže [82], stabla odlučivanja [40] [51], metoda nasumične šume [51], jednostavni Bayesov klasifikator [84], metode nadopunjavanja slabih klasifikatora (engl. *boosting*) [40] [95] i drugi. U istraživanjima često isti algoritmi postižu različite rezultate. U nadziranom učenju taj fenomen je poznat kao iskaz o "nepostojanju besplatnog ručka" (engl. *no free lunch theorem*) koji govori da ne postoji univerzalni algoritam koji postiže najbolje rezultate za svaki problem. Osnovni problem kod detekcije zlonamjernih programa je taj da ne postoji standardni skup podataka koji se koristi za učenje modela, već je na istraživaču da od dostupnih uzoraka programa stvori skup za učenje i testiranje.

Zlonamjerni programi sadrže vremensku komponentu pa je s vremenom potrebno nadopuniti skup za učenje novim varijacijama zlonamjernih programa i model ponovno naučiti. Upravo zbog kvalitete izabranih podataka, tj. programa, iste metode strojnog učenja u istraživanjima postižu različite rezultate. Prema trenutačnom stanju znanosti, za dani problem najbolje rezultate (općenita točnost>90%) postižu metode potpornih vektora, nasumične šume i nadopunjavanje slabih klasifikatora. Dakle, metode strojnog učenja predstavljaju alate koji pomažu pri automatskoj klasifikaciji, ali pored metoda za dobar klasifikacijski model potrebno je imati i kvalitetne značajke koje opisuju uzorke programa.

2.4.1 Klasifikacijske metode iz domene strojnog učenja

Područje strojnog učenja posjeduje metode koje omogućuju efikasno rješavanje klasifikacijskih problema, kao što je u ovom slučaju klasifikacija u klase benignih ili malicioznih programa. Postoji širok niz metoda i njihovih varijacija koje je moguće primijeniti na ovaj problem, ali za odabir odgovarajućeg osnovnog klasifikatora (0-reda) odabrani su algoritmi koji postižu dobre rezultate u dosadašnjoj praksi i sličnim istraživanjima. Svaki od osnovnih klasifikatora može u konačnom hibridnom klasifikatoru koristiti jednu od sljedećih metoda: jednostavni Bayesov klasifikator, logistička regresija, stablo odlučivanja C 4.5, metoda potpornih vektora SVM ili metoda nasumične šume. Prilikom postupka učenja, unakrsnom validacijom određujemo najbolji klasifikator na temelju rezultata nakon svih preklapanja, a u ovom istraživanju koristi se 10 preklapanja koji predstavlja uobičajeni izbor broja preklapanja zbog smanjenja moguće varijance rezultata između svakog preklapanja.

Na odabir metoda, osim dostupnosti u praksi i drugim istraživanjima utjecala su i njihova svojstva, primjerice uspoređujemo jednostavnije i vremenski nezahtjevne metode poput logističke regresije i jednostavnog Bayesovog klasifikatora. Stablo odlučivanja C 4.5 je

metoda koja daje razumljiva pravila kojima je moguće objasniti naučene koncepte i koja daju uvid u način donošenja klasifikacije svake pojedine instance, dok SVM ima mogućnost promjene koordinatnog sustava koristeći različite jezgrene funkcije za teško rješive klasifikacijske probleme. Metoda nasumične šume omogućuje korištenje većeg broja stabla odlučivanja u klasifikaciji kako bi se poboljšala točnost rezultata, ali zbog tog svojstva metoda ne posjeduje odgovarajuću interpretabilnost modela kao što je slučaj kod metode stabla odlučivanja.

U sljedećem dijelu dan je kratki opis korištenih metoda.

Bayesov klasifikator je vjerojatnosni klasifikator koji za klasifikaciju koristi Bayesov teorem i oslanja se na pretpostavku neovisnosti između značajki [63]. Učenje se provodi nad skupom kojeg opisuju značajke $\{a_1, a_2, \cdots, a_n\}$ i označen je klasama skupa V. Nakon faze učenja na skupu za učenje naučeni koncept predviđa klasifikaciju novih instanci. Nad danim značajkama $\{a_1, a_2, \cdots, a_n\}$ računa se najvjerojatnija vrijednost v_{MAP}:

$$v_{MAP} = \underset{v_j \in V}{\operatorname{argmax}} P(v_j | a_1, a_2, \cdots, a_n) \tag{2.1}$$

Možemo primijetiti kako se gornji izraz oslanja na Bayesov teorem:

$$P(v_j | a_1, a_2, \cdots, a_n) = \frac{P(a_1, a_2, \ldots, a_n | v_j) P(v_j)}{P(a_1, a_2, \cdots, a_n)} \tag{2.2}$$

Zato što se jednostavni Bayesov klasifikator oslanja na neovisnost između značajki u odnosu na klasifikacijsku vrijednost, izbacuje se nazivnik odnosno umnožak vjerojatnosti a_i.

Izraz možemo zapisati i kao:

$$v_{NaiveBayes} = \underset{v_j \in V}{\operatorname{argmax}} P(v_j) \prod_i P(a_i | v_i) \tag{2.3}$$

Učenje jednostavnog Bayesovg klasifikatora uključuje računanje vjerojatnosti pojavljivanja klase $P(v_j)$ u odnosu na dane značajke $P(a_i|v_j)$. Skup ovih procjena čine naučenu hipotezu, koja se kasnije koristi za klasificiranje novih primjera na korištenjem izraza (2.3).

Logistička regresija dodjeljuje određenu vjerojatnost pojedinim klasifikacijskim vrijednostima iz skupa $C = \{C_1, .., C_n\}$ [41]. Kako bi procijenili vjerojatnost pripadnosti klasama koriste se značajke $X = \{x_1, .., x_n\}$ i logistička (sigmoidalna) funkcija koja preslikava realne brojeve u interval $[0, 1]$ te je definirana sljedećim izrazom:

$$h(X) = \frac{1}{1 + e^{\beta_0 + \beta_1 x_1 + .. + \beta_n x_n}} \tag{2.4}$$

koju možemo zapisati kao:

$$p(X) = \frac{e^{\beta_0+\beta_1 x_1+..+\beta_n x_n}}{1+e^{\beta_0+\beta_1 x_1+..+\beta_n x_n}} \tag{2.5}$$

ili

$$\frac{p(X)}{1-p(X)} = e^{\beta_0+\beta_1 x_1+..+\beta_n x_n} \tag{2.6}$$

Lijeva strana prethodnog izraza $p(X)/1-p(X)$ naziva se još i izglednost (engl. *odds*) i može poprimiti vrijednosti od 0 do ∞. Logaritmiranjem po bazi e dobivamo logaritamsku izglednost ili logit :

$$log(\frac{p(X)}{1-p(X)}) = \beta_0 + \beta_1 x_1 + .. + \beta_n x_n \tag{2.7}$$

Učenje parametra logističke regresije $\beta_0, .., \beta_n$, kako bi svakom primjeru pridodali klasu iz skupa klasa C, svodi se na minimiziranje funkcije pogreške, koja se može definirati kao negativna log-izglednost:

$$E(h|C) = E(\beta_0, .., \beta n|C) = -\log l(\beta_0, .., \beta n|C) \tag{2.8}$$

U tom slučaju je minimizacija funkcije pogreške jednaka maksimizaciju log-izglednosti (engl. *maximum likelihood estimation*) i predstavljena je funkcijom [41]:

$$l(\beta_0, .., \beta n) = \prod_{i=1:y_i=1}^{N} p(x_i) + \prod_{i=1:y_i=0}^{N} (1-p(x_i)) \tag{2.9}$$

ili

$$l(\beta_0, .., \beta n) = \sum_{i=1:y_i=1}^{N} \log p(x) + \sum_{i=1:y_i=0}^{N} \log (1-p(x_i)) \tag{2.10}$$

Za maksimizaciju log-izglednosti može se koristi iterativna optimizacijska metoda gradijentnog spusta. Procijenjeni parametri $\beta_0, .., \beta_n$ koji maksimiziraju funkciju log-izglednosti, odnosno minimiziraju funkciju pogreške, uvrštavaju se u izraz (2.5). Na temelju naučene granice $y = \beta_0 + \beta_1 x_1 + .. + \beta_n x_n$ dodjeljuju se klase klasifikacijskog problema za svaki primjer koji je potrebno klasificirati.

Stablo odlučivanja (algoritam C 4.5) je nadogradnja klasičnoga algoritma ID3, oba algoritma su rezultat istraživanja Rossa Quinlana [63]. Stablo odlučivanja predstavlja graf u kojem su čvorovi značajke, a lukovi njihove vrijednosti ili intervali numeričkih vrijednosti [13]. C 4.5 prvo stvara od skupa za učenje prenaučeno (redundantno) stablo, u slučaju klasifikacije sa podcima sličnima s onima iz skupa za učenje, klasifikator postiže dobre rezultate dok na podacima iz neovisnog skupa mogući su lošiji rezultati. Kao mjere kvalitete razdvajanja koriste se informacijski dobitak (engl. *information gain*) ili Gini indeks. Na temelju najveće mjere kvalitete razdvajanja rekurzivno se odabiru značajke

koje se stavljaju u korijen stabla ili podstabla.

Za skup svih primjera S i skup svih značajki F, gdje je S_f broj primjera iz skupa S sa značajkom f, mjera informacijskog dobita $InfoGain$ se definira kao: [58]:

$$InfoGain(S, F) = Entropija(S) - \sum_{f \in F} \frac{S_f}{S} Entropija(S_f) \tag{2.11}$$

gdje je $Entropija(S)$ za svaku primjer s klasom koja pripada skupu C:

$$Entropija(S) = -\sum_{i \in C} p_i log_2 p_i \tag{2.12}$$

Alternativno, može se koristiti i mjera razdvajanja Gini indeks [58], gdje U $N(i)$ predstavlja vjerojatnost klasifikacije primjera koji pripada klasi i :

$$G_k = 1 - \sum_{i=1}^{c} N(i)^2 \tag{2.13}$$

Gini indeks predstavlja "nečistoću" skupa, odnosno "čisti" čvor svaki primjer označuje s istom klasom te ima vrijednost indeksa $Gini = 0$ dok će "nečisti" čvor imati jednak broj klasifikacija po svim klasama ($Gini = 0.5$).

Kako bi se spriječilo pretreniranje, na redundatno stablo se primjenjuje postupak podrezivanja (engl. *pruning*). Stablom se induciraju Ako/Onda pravila te se računaju uvjeti kojima se postiže najveća točnost klasifikacije, a na način da se izbacuju podstabla koja ne poboljšavaju točnost klasifikacije. Podrezivanje nakon izgradnje redundatnog stabla počinje od listova prema korijenu stabla te je temeljno na pesimističnoj procjeni grešaka, koja se odnosi na postotak krivo klasificiranih slučajeva u skupu za učenje. Nakon podrezivanja listovi se označuju onom klasom koja prevladava u danim primjerima [92]. Na temelju razlike u točnosti pravila i standardne devijacije uzete iz binomne razdiobe uzima se gornja granica povjerenja (engl. *confidence*) koja najčešće iznosi 0.25, a na temelju koje se podstabla podrezuju. Druga mogućnost podrezivanja stabla je za vrijeme njegove izrade, na način da se ograničava rast stabla ako je broj primjera premali za donošenje konačne odluke.

Prednosti algoritma C 4.5 za indukciju stabla odlučivanja u odnosu na njegovog prethodnika, algoritam ID3, jesu [63]:

- C 4.5 može koristiti atribute s numeričkim vrijednostima (kontinuirane vrijednosti) osim diskretnih značajki. Kontinuirani atributi se pretvaraju u skup diskretnih vrijednost koji dijele cjelokupni skup na temelju pragova.

- C 4.5 stablo se podrezuje nakon izgradnje; kako bi izbacila suvišna podstabla. S ovom metodom se smanjuje pretreniranost klasifikatora.
- C 4.5 funkcionira s nedostajućim vrijednostima
- C 4.5 omogućuje normalizaciju značajki s različitim težinskim faktorima.

Metoda potpornih vektora (engl. Support Vector Machine) predstavlja algoritam koji pronalazi najveću marginu razdvajanja između klasa, a pod marginom podrazumijevamo udaljenost koja obuhvaća kritične primjere, tj. primjere koji bi bili krivo klasificirani da ne koristimo marginu već samo plohu razdvajanja.

Primjere najbliže plohi razdvajanja nazivamo potpornim vektorima, a margina M predstavlja udaljenost od plohe razdvajanja (prikazano na slici 2.2).

Računanje potpornih vektora je optimizacijski problem [41]:

$$max_{\beta 0,..,\beta n,\xi 2,..,\xi n} M \tag{2.14}$$

u odnosu na

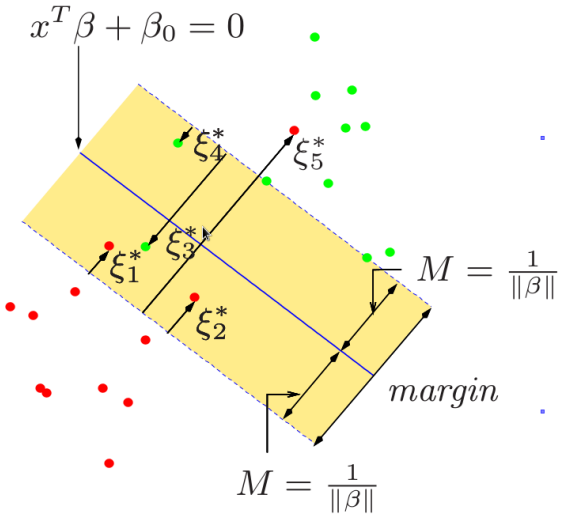
$$\sum_{j=1}^{n} \beta_j^2 = 1 \tag{2.15}$$

$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip}) \geq M(1 - \xi_i) \tag{2.16}$$

uz uvjete

$$\xi_i \geq 0, \sum_{i=1}^{n} \xi_i \leq C \tag{2.17}$$

ξ_i nam pokazuje na kojoj strani margine je opservacija i, za $\xi_i = 0$ opservacija je na ispravnoj strani dok za $\xi i > 0$ je na krivoj strani. Vrijednost $\xi_i > 1$ nam govori da se opservacija nalazi na krivoj strani hiperravnine. Parametar C nam govori o broju opservacija koje su na krivoj strani margine ili hiper ravnine, C se izabire u postupku unakrsne validacije i njime se kontrolira odnos između pristranosti i varijance (engl. bias-variance tradeoff). U pravilu, kada je C manji margina je bliža odnosno instance se nastoje klasificirati što točnije. Dok u slučaju kada je C veći margina je šira i omogućava se veći broj krivo klasificiranih instanci što daje modelu veću pristranost, ali uz manju varijancu.



Slika 2.2: Klasifikacija potpornim vektorima za slučaj preklapanja - margina je označena s M dok ξ_j^* označava instance na krivoj strani margine (preuzeto iz [34])

Trik koji se koristi kod SVM-a je korištenje različitih jezgrenih funkcija, koje neseparabilan problem premještaju u višu dimenziju gdje je problem pogodniji za rješavanje.

U ovom istraživanju korištena je linearna jezgrena funkcija, polinomna jezgrena funkcija drugog i trećeg stupnja te radijalna jezgrena funkcija. Jezgrena funkcija je unutarnji produkt koji kvantificira sličnost između dvije opservacije (x_i i $x_{i'}$) [41]. Kod linearne jezgrene funkcije unutarnji produkt kvantificira sličnost koristeći Pearsonovu korelaciju te se izračuna izrazom:

$$K(x_i, x_{i'}) = \sum_{j=1}^{n} x_{ij} x_{i'j} \tag{2.18}$$

Polinomnu jezgrena funkciju računamo sljedećim izrazom:

$$K(x_i, x_{i'}) = (1 + \sum_{j=1}^{n} x_{ij} x_{i'j})^d, d > 1 \tag{2.19}$$

Dok, radijalnu jezgrenu funkciju računamo na sljedeći način:

$$K(x_i, x_{i'}) = e^(- \gamma \sum_{j=1}^{n} (x_{ij} x_{i'j})^2), \gamma > 0 \tag{2.20}$$

Metoda nasumične šume (engl. Random Forest) koristi veći broj nezavisnih stabla odlučivanja kako bi smanjila varijancu konačnog modela, a slična je metodi *bagging* od koje se razlikuje po postupku dekorelacije. Skup za učenje B se podijeli na podskupove s jednakim brojem primjera, nakon toga se od svakog podskupa napravi stablo odlučivanja. Pošto je broj podataka u skupu za učenje ponekad ograničen prilikom odabira primjera za podskupove koristi se metoda nasumičnog uzrokovanja s vraćanjem primjera (engl.

bootstrap). Kako bi se smanjila varijanca konačni model je kod regresijskih problema aritmetička sredina svih naučenih stabla odlučivanja, ili kod klasifikacijskih problema većinska klasa dobivena metodom glasovanja. U metodi nasumične šume svako stablo se izgrađuje iz N različitih *bootstrap* uzorka izuzetih iz početnog skupa za učenje, a trećina izdvojenih primjera se koristi za testiranje izgrađenog modela nasumične šume tj. procjenu pogreške (engl. *out of bag error*). Broj stabla odlučivanja se povećava dok se točnost ne ustali, ili se vodi računa o broju stabla odlučivanja koji se definira kao parametar metode.

Metoda nasumične šume se razlikuje od metode *bagginga* po tome što algoritam vodi računa da se koristi samo određeni broj nasumično izabranih značajki (najčešće $\sqrt{p}$) prilikom stvaranja svakog pojedinačnog čvora u stabla odlučivanja, kao što je opisano u izvornom istraživanju Breimana [18]. Na taj način unosi se raznolikost jer se odabiru podskupovi s različitim primjerima i značajkama. Povećanjem broja značajki povećava se točnost klasifikacije pojedinog stabla ali i međusobna korelacija između stabla, što smanjuje konačnu točnost klasifikacije šume. Ovaj postupak se još naziva dekorelacija [41], kojom se izbjegava korištenje koreliranih primjera i time smanjuje prenaučenost (engl. *overfitting*) modela. U pravilu se u konačnom modelu odabiru značajke koje omogućuju najbolje račvanje pojedinih čvorova, odnosno odabiru se oni čvorovi s najvećom vrijednošću Gini indeksa ili informacijskog dobitka.

2.4.2 Mjere uspješnosti u kontekstu predloženog pristupa

Binarni klasifikaciji problem izbora prirode programa sadrži dvije klase - zlonamjerne programe i dobronamjerne programe. U našem slučaju pozitiv predstavlja zlonamjerni program. Ukoliko klasifikator ispravno prepoznaje zlonamjerni program definiramo ga kao **istiniti pozitiv** (engl. *true positive* - TP). Ukoliko klasifikator dobronamjerni program označi zlonamjernim, takav slučaj zovemo **lažni pozitiv** (engl. *false postive* - FP). Dakle, FP je lažno upozorenje da je program zlonamjeran. Dobronamjerni program je u našem slučaju negativ. Kada klasifikator ispravno prepozna dobronamjerni program tada se ta klasifikacija naziva **istiniti negativ** (engl. *true negative* - TN). Nasuprot tome, kada klasifikator javi da je program dobronamjeran, a on je u stvarnosti zlonamjeran, tada taj slučaj zovemo **lažni negativ** (engl. *false negative* - FN).

Stopu istinitih pozitiva ili osjetljivost[1] (engl. *True postive rate* TPR, *Sensitivity*) definiramo kao:

$$TPR = \frac{TP}{TP + FN} \tag{2.21}$$

Idealan klasifikator ima stopu istinitih pozitiva $TPR = 1$. TPR je omjer između uspješno

[1] U prirodnoj obradi teksta ova mjera se još naziva odaziv (engl. recall)

klasificiranih zlonamjernih programa i svih programa koji su označeni kao zlonamjerni $(TP + FN)$.

Stopu lažnih pozitiva (engl.*False Positive Rate* - FPR) definiramo kao:

$$FPR = \frac{FP}{FP + TN} \tag{2.22}$$

FPR je omjer između pogrešno označenih zlonamjernih programa i ukupnog broja klasificiranih dobronamjernih programa. Na sličan način definiramo mjeru TNR, koju zovemo još i specifičnost (engl. *specificity*) i FNR:

$$TNR = \frac{TN}{TN + FP} \tag{2.23}$$

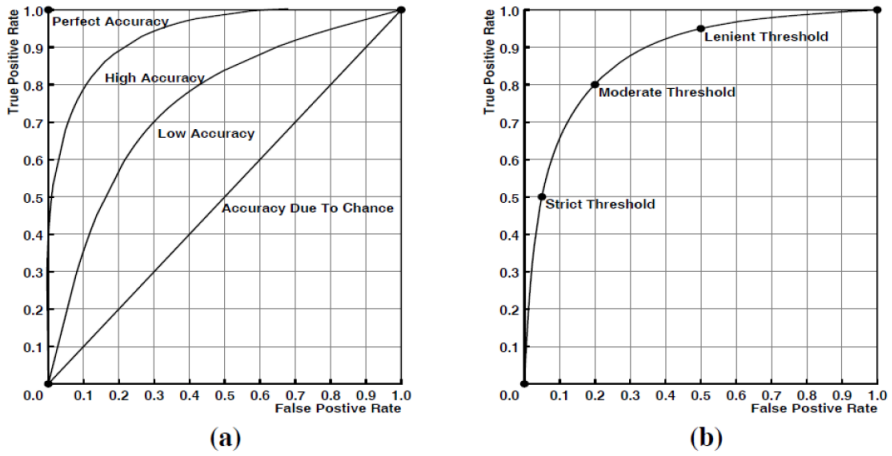
$$FNR = \frac{FN}{FN + TP} \tag{2.24}$$

Općenita točnost klasifikacije (engl. *accuracy*) je omjer koji prikazuje ukupnu točnost klasifikatora, a definira se kao:

$$Acc = \frac{(TP + TN)}{(TP + TN + FP + FN)} \tag{2.25}$$

Preciznost klasifikacije definiramo kao:

$$Prec = \frac{TP}{TP + FP} \tag{2.26}$$



Slika 2.3: Primjer ROC krivulje (preuzeto iz [60])

ROC krivulja (engl. *Receiver operating characteristic curve*) često se koristi za procjenu uspješnosti klasifikatora. ROC krivulja je dvodimenzionalni prikaz točnosti detekcije određene klase. U našem slučaju na y-osi se nalazi stopa ispravnih pozitiva (TPR) dok se na x-osi nalazi stopa lažnih pozitiva, odnosno pogrešno označenih zlonamjernih programa

(FPR). Krivulja idealanog klasifikatora prolazi kroz točku (0,1) dok je krivulja nasumičnog klasifikatora predstavljena pravcem (prikazano na slici 2.3.a). Kod zadovoljavajućeg klasifikatora stopa TPR treba rasti brže nego stopa FPR (prikazano na slici 2.3.b). Kao sumarna statistika kod ROC krivulje koristi se površina ispod krivulje (engl. *area under the curve* ili AUC). Nasumični klasifikator imat će prema tome $AUC = 0.5$, dok će klasifikator s boljom točnošću od nasumičnog pogađanja imati $AUC > 0.5$.

2.4.3 Metode za odabir značajki i redukciju dimenzionalnosti

Često podaci kod klasifikacijskih problema pate od problema velike dimenzionalnosti, odnosno problem koji se rješava opisan je velikim brojem značajki koje su vrlo različite, a neke od njih nisu dovoljno informativne već samo odmažu pri klasifikaciji. Velika dimenzionalnost može biti uzrok spore brzine učenja modela, ali može utjecati i na problem pretreniranja (engl. *overfitting*) zbog podešavanja parametra metode vrijednostima bliskim skupu za učenje. Pretrenirenje modela se očituje u slabijim rezultatima koje naučeni model postiže na novim (nepoznatim) primjerima, npr. u testnom skupu, u odnosu na rezultate koje model postiže na skupu za učenje. Iz tog razloga čest je slučaj da naučeni model postaje složen za dani skup za učenje te postoji velika razlika (varijanca) u rezultatima između skupa za učenje i testiranje. Kako bi smanjili ovu pojavu moguće je primijeniti sljedeće korake [75]:

- Ukoliko je moguće proširiti skup za učenje novim primjerima;
- Koristiti regularizaciju za klasifikacijski algoritam kao penalizacijski mehanizam;
- Izabrati jednostavniji (reducirani) skup značajki;
- Reducirati dimenzionalnost podataka;

U ovom istraživanju korištena je metoda glavnih komponenata koja služi za redukciju dimenzionalnosti početnog skupa te metoda rekurzivne eliminacije za odabir optimalnog podskupa značajki.

Metoda glavnih komponenata (eng. *principal component analysis* ili PCA) je nenadzirana metoda strojnog učenja koja se koristi za redukciju dimenzionalnosti početnog skupa podataka. Glavne komponente predstavljaju nove značajke odnosno linearne kombinacije početnog skupa. Cilj metode glavnih komponenta je pronalazak novog koordinatnog sustava, s manjim brojem dimenzija, koji sadržava varijabilnost početnog skupa. Glavne komponente zadržavaju varijabilnosti podataka, te su međusobno ortogonalne. Prva glavna komponenta predstavlja smjer duž kojeg je najveća varijanca početnog skupa, druga glavna komponenta je smjer s najvećom varijancom koja je ortogonalna na prvu glavnu komponentu, treća glavna komponenta je smjer s najvećom varijancom ortogonalan

na drugu glavnu komponentu, itd.

Početni skup podataka predstavljen je matricom X s n redaka koje predstavljaju opažanja (primjera) problema i d stupaca odnosno značajki. Analizom glavnih komponenta dobivamo novu matricu W dimenzija $n \times r$ gdje je $r < d$.

Za primjenu metode glavnih komponenata nije potrebno zadovoljiti prepostavke homogenosti varijance ili pripadanja podataka normalnoj razdiobi, ali je poželjno standardizirati podatke tako da imaju srednju vrijednost $\mu = 0$ i standardnu devijaciju $\sigma = 1$ [89]. Standardizacija se primjenjuje iz razloga što svim značajkama želimo dati jednaku važnost. Naime, često se događa da su značajke mjerene na različitim skalama i neke imaju veću varijancu od ostalih što u konačnici loše utječe pri redukciji dimenzija zbog pridavanja veće važnosti značajkama s takvim obilježjem. Nakon standardizacije, računa se matrica kovarijacija Σ dimenzija $d \times d$, koja sadrži usporedne kovarijacije između značajki matrice X.

Primjerice, kovarijanca između vrijednosti dvije značajke X_j i X_k jednaka je:

$$\sigma_{j,k} = \frac{1}{n} \sum_{i=1}^{n} (x_j^{(i)} - \mu_j)(x_k^{(i)} - \mu_k) \tag{2.27}$$

Gdje su μ_j i μ_k aritmetičke sredine značajke (stupca) j, odnosno k. Pozitivna kovarijanca između vrijednosti dvije značajke označava da su promjene vrijednosti povezane, tj. rast jedne značajke utjecat će na rast druge značajke. Kod negativne kovarijance promjene između značajki djeluju u suprotnosti, povećanje jedne značajke označava pad vrijednosti druge značajke i obrnuto.

Konačna matrica kovarijanci Σ ima sljedeći oblik, za dimenziju matrice 3×3 :

$$\Sigma = \begin{bmatrix} \sigma_1^2 & \sigma_{12} & \sigma_{13} \\ \sigma_{21} & \sigma_2^2 & \sigma_{23} \\ \sigma_{31} & \sigma_{32} & \sigma_3^2 \end{bmatrix} \tag{2.28}$$

Novi, reducirani, koordinatni sustav razapet je svojstvenim vektorima najvećih svojstvenih vrijednosti kovarijacione matrice skupa podataka [14]. Svojstveni vektori matrice Σ predstavljaju glavne komponente, odnosno smjer s najvećom varijancom, dok svojstvene vrijednosti njihovu veličinu. Svojstvene vrijednosti λ definirane su kao nul-točke jednadžbe:

$$det(\lambda I - \Sigma) = 0 \tag{2.29}$$

Dok svojstveni vektori (v) za kvadratnu matricu kovarijanci se dobivaju iz izraza:

$$\Sigma v = \lambda v \tag{2.30}$$

Za dobivenih d svojstvenih vrijednosti i svojstvenih vektora matrice Σ vrijede sljedeća svojstva [14]:

- Svakom svojstvenom vektoru odgovora jedna svojstvena vrijednost
- Jednoj svojstvenoj vrijednosti može odgovarati beskonačno mnogo svojstvenih vektora
- Svakoj svojstvenoj vrijednosti pripada samo jedan jedinični svojstveni vektor
- Svojstveni vektori koji pripadaju različitim svojstvenim vrijednostima međusobno su ortogonalni

Kako bi transformirali početni skup X u konačni skup W potrebno je odabrati vodećih r jediničnih svojstvenih vektora kovarijacijske matrice Σ. Pod pojmom vodeći podrazumijevaju se svojstveni vektori s najvećim svojstvenim vrijednostima. Izabranih r vodećih svojstvenih vektora čine matricu V, dimenzija $d \times r$, koju koristimo za projiciranje u prostor manje dimenzionalnosti. Matrica W predstavlja projekciju iz izvornog prostora dimenzije d, kojeg predstavlja matrica X, u novi prostor dimenzije r:

$$W^T_{(r\times n)} = V^T_{(r\times d)} X^T_{(d\times n)} \tag{2.31}$$

Dimenziju novog prostora omeđenog s r svojstvenih vektora izabiremo na način da projekcijom početnog skupa u novi prostor gubimo što manje informacija. Gubljenje informacija odnosno varijabilnosti početnog skupa određeno je svojstvenim vrijednostima, stoga cilj nam je odabrati r najvećih svojstvenih vrijednosti koji zadržavaju najviše varijance početnog skupa. Za računanje zadržane varijance početnog skupa odabirom r svojstvenih vrijednosti λ koristimo sljedeći izraz [75]:

$$Vr = \frac{\sum_{j=1}^{r} \lambda_j}{\sum_{j=1}^{d} \lambda_j} \tag{2.32}$$

Metoda rekurzivne eliminacije (engl. *recursive feature elimitation*, RFE) značajki služi za odabir značajki s kojima korišteni klasifikator postiže rezultate koji su usporedivi s korištenjem cijelog skupa značajki. Metoda rekurzivne eliminacije pripada skupini metoda koje koriste omotač (engl. *wrapper*) za odabir značajki [45]. Pod omotačem smatra se nadzirani klasifikaciji algoritam, npr. SVM [33], a s kojim metoda traži podskup značajki s prihvatljivom točnošću u odnosu na cjeloviti skup značajki. Metode omotača mogu dodavati ili odbacivati značajke kako bi došle do optimalnog podskupa značajki.

Metoda rekurzivne eliminacije kreće od cjelovitog skupa značajki te postepeno odbacuje značajke koje ne pridonose klasifikaciji. Guyon je prva predstavila metodu rekurzivne eliminacije, a za procjenu točnosti s različitim podskupovima značajki koristila je SVM [33]. Postoji varijacija koja koristi metodu nasumične šume za rekurzivnu eliminaciju [31], a za mjeru rangiranja koristi se prosječna greška klasifikacije instanci iz skupa za učenje koje nisu prethodno izuzete uzrokovanjem (engl. *out of bag error*). Pseudokod algoritam rekurzivne eliminacije prikazan je u izlistu Algoritam 1. Algoritam RFE svodi se na izračun rangirane liste značajki r, gdje se u svakoj iteraciji koristi podskup značajki s, u kojem je izbačena najslabije rangirana značajka s liste r.

Skup za učenje $X_0 = [x_1, .., x_k, .., x_l]^T$;
Oznake klasa $y = [y_1, .., y_k, .., y_l]^T$;
Podskup značajki $s = [1, 2, .., n]$;
Rangirane značajke $r = []$;
ponavljaj
    Koristi samo izabrane značajke $X = X_0(:, s)$;
    Nauči klasifikator $\alpha = SVM(X, y)$;
    Izračunaj težinski vektor duljine s, $w = \sum_k \alpha_k y_k x_k$;
    Za svaku značajku i računamo kriterij važnosti značajki $c_i = (w_i)^2$;
    Pronađi najslabije rangiranu značajku $f = argmin(c)$;
    Ažuriraj listu rangiranih značajki r;
    Izbaci najslabije rangiranu značajku iz skupa s;
dok ne bude $s = []$;
vrati listu rangiranih značajki r

Algoritam 1: Metoda rekurzivne eliminacije (RFE) prema [33]

Poglavlje 3

Pregled srodnih istraživanja

Maliciozni program je općeniti naziv za programsku komponentu koja izvodi malicioznu namjeru njegovog autora. S vremenom maliciozni programi postali su sve napredniji te se njihovim razvojem razvijaju i mjere zaštite u sigurnosnim alatima. Automatski sustavi za detekciju malicioznih programa predstavljaju sustave koji mogu prepoznati maliciozne programe bez ljudske intervencije. Neki od njih nude mogućnost uklanjanja malicioznog sadržaja. Najpoznatija zaštita od malicioznih sadržaja u posljednjih dvadesetak godina su antivirusni alati. Njihov detekcijski mehanizam temelji se na potpisima (engl. *signatures*) i heuristikama. Analitičari zaposleni u sigurnosnim tvrtkama analizom malicioznih programa stvaraju potpise, odnosno obrasce koji prepoznaju maliciozne programe. Slično je i s heurističkim metodama, koje predstavljaju pravila kojima se otkrivaju općenite varijante malicioznih programa. Tvrtke koje proizvode sigurnosne alate ne otkrivaju svoje metode detekcije malicioznih sadržaja. Tako se samo neki alati opredjeljuju za potpise uz korištenje heuristika kao pomoćnog mehanizma zaštite. Metode detekcije sigurnosnih alata nisu imune na greške pa su stoga poznati slučajevi sigurnosnih alata koji prepoznaju svoje komponente kao maliciozne [98] [80].

Nasuprot sigurnosnim tvrtkama stoje autori malicioznih programa kojima je cilj zaobilaženje sigurnosnih mehanizama. Autori malicioznih programa su najčešće organizirani kriminalnci koje motivira financijska korist te u novije vrijeme države koje maliciozne programe koriste u svrhu ratovanja [57]. Njihov način razmišljanja i metode razvoja uvelike se razlikuju od programera. Oni traže propuste operacijskog sustava i alata kako bi zarazili računalni sustav i nanijeli štetu žrtvi. Najjednostavniji način zaobilaženja sigurnosne zaštite je sažimanje ili pakiranje malicioznog programa. Postoje i naprednije metode poput polimorfnih malicioznih programa koji mijenjaju svoj oblik i mogu imati na desetke ili stotine tisuća varijacija. Autori malicioznih programa ovim metodama otežavaju detekciju sigurnosnim alatima. Potpisi su u potpunosti neučinkoviti za prepoznavanje polimorfnih i zapakiranih programa. Često je potrebno nekoliko dana prije nego što analitičari naprave analizu malicioznog programa i izrade nove potpise za njihovu detekciju.

Kako bi se riješili spomenuti problemi istraživači su počeli razvijati i istraživati nove metode za automatsku klasifikaciju malicioznih programa. Posebno su popularne metode strojnog učenja koje omogućavaju brzo prepoznavanje obrazaca koje koriste maliciozni programi i koji ih razlikuju od benignih programa [76] [87] [93]. Značajke koje opisuju program neophodne su za podešavanje metoda strojnog učenja te o njima uvelike ovisi uspješnost klasifikacije malicioznih programa. Njih je moguće dobiti metodama analize izvršnih datoteka. Pristupi koji se koriste pri analizi programa su statička i dinamička analiza. Statičkom analizom utvrđuje se što program radi promatranjem tijeka i grananja njegovom disasembliranog ekvivalenta. Dinamičkom analizom program se izvodi kako bi se utvrdilo njegovo ponašanje i način rada. Često se rezultati statičke i dinamičke analize nadopunjuju i omogućuju bolje razumijevanje namjere malicioznih programa.

U ovom poglavlju prikazane su korištene metoda analize i značajke izvršnih programa u trenutačnom stanju znanosti. Također, dan je pregled metoda strojnog učenja koje se koriste u relevantnim istraživanjima te njihova uspješnost.

3.1 Mogućnosti analize zlonamjernih programa

Analitičari koriste dvije osnovne vrste analize prilikom disekcije malicioznog sadržaja: statičku i dinamičku analizu.

Statička analiza podrazumijeva prebacivanje izvršne datoteke u jezik najniže razine, Asembler, te analizu kôda najniže razine bez njegovog izvršavanja. Cilj statičke analize je utvrditi što zlonamjerni program radi i ukoliko je moguće prepoznati obrazac koji ima malicioznu namjeru kako bi se on mogao uključiti u potpis. Postoji više razina statičke analize i prikaza promatranih podataka u njima pa tako korištenje statičke analize u trenutnom stanju znanosti možemo podijeliti na sljedeće metode:

a) *Analiza strukture i vrijednosti unutar izvršne datoteke* temelji se na analizi strukture *Poratble Executable* (PE) datoteke, tj. direktnim uzimanjem vrijednosti atributa iz zaglavlja i sekcija ili izvođenjem novih atributa iz postojećih. Ovaj pristup koriste sustav IMDS [106] i istraživanje Schultza [84] koji iz PE datoteke izvlače uvezene i izvezene biblioteke funkcija. Pošto se PE format sastoji od sekcija moguće je izračunati entropiju svake sekcije i utvrditi je li program pakiran. Ovaj pristup koriste Perdisci i suradnici [69].

b) *Analiza izvršne (disasemblirane) datoteke* - postoje dvije vrste prikaza izvršnog kôda koje je moguće dobiti iz kompajlirane izvršne datoteke - asemblerske instrukcije i strojni kôd.
Strojni kôd ili bajt-kôd (engl. *machine code*) je ekvivalentan asemblerskim instrukcijama, ali je prikazan u nižem obliku - zbog preglednosti heksadecimalnom obliku. Primjer alata za dobivanje strojnog kôda su alati za prikaz heksadecimalnog bajt-kôda (hex-editori) poput preglednika Hiew ili HxD te alati koji pretvaraju izvršnu datoteku u bajt-kôd poput Hexdumpa. Strojni kôd sadrži pojedine instrukcije nazvane operacijski kôd (engl. *operation code*) koje su zadužene za izvršavanje operacija nad operandima, primjerice vrijednostima u registrima ili stogu. Operacijski kôdovi se tako primjerice mogu koristiti za računanja entropije izvršne datoteke [69]. Asemblerske instrukcije i operacijski kôdovi su međusobno povezani na način da je operacijski kôd heksadecimalni prikaz asemblerske instrukcije. Primjerice, skok JMP u asembleru ima operacijski kôd 0xE9[1] na Intel x86 arhitekturi. Alazab i suradnici [3] koriste analizu operacijskih kôdova na način da uzimaju u obzir učestalost poziva prema operacijskim kôdovima CALL i JMP, koji služe za pozivanje modula i skokove, prema kojima razlikuju maliciozne programa od onih benignih.

[1] x86 Opcode Structure and Instruction Overview https://net.cs.uni-bonn.de/fileadmin/user_upload/plohmann/x86_opcode_structure_and_instruction_overview.pdf

Sustav EUREKA [88] stvara bi-grame od dvo-bajtnih operacijskih kôdova te dobivene bi-grame testiraju z-testom kako bi prepoznali pakirane izvršne datoteke. Operacijski kôdovi se još koriste [7] za generiranje pojednostavljenih grafova tijeka podataka. U sustavu OPEM [82] operacijski kôdovi se koriste kao značajke za klasifikaciju izvršnih datoteka u odnosu na ponderiranu učestalost pojavljivanja poziva prema operacijskom sustavu. Oblici niže razine izvršnih datoteka, poput strojnog kôda i operacijskih kôdova, u kombinaciji s n-gram modelima, korišteni su za klasifikaciju izvršnih datoteka i u drugim istraživanjima [1], [52] i [96].

c) *Graf kontrole tijeka podataka i graf poziva među funkcijama* (engl. *control flow graph*, CFG) je grafički prikaz koji prikazuje sve tijekove izvršavanja koje program može poprimati u svom životnom ciklusu. CFG je usmjereni graf $C = (B, E)$ predstavljen blokovima slijednih instrukcija B i grananjima (skokovima) između njih E. CFG predstavlja petlje, uvjetna i neuvjetna grananja i njihovu povezanost sa slijednim funkcijama. Ovakva vrsta prikaza uvelike olakšava analizu analitičarima i prikazuje sve moguće tijekove kroz pojedinu proceduru. Graf poziva među funkcijama $C = (F, E)$ je graf koji predstavlja moguće pozive E prema funkcijama F korištenim u programu. Graf poziva je usmjeren te može biti aciklički ukoliko ne postoje rekurzivne procedure u programu. Razlika između grafa poziva među funkcijama i grafa kontrole tijeka podataka je u razini detaljnosti. Graf poziva između funkcija prikazuje grananja u odnosu na funkcije i njihovu povezanost, dok CFG prikazuje i što se događa u samim funkcijama. Alat IdaPro [35] je u mogućnosti grafički prikazati dobiveni disasemblirani ekvivalent u obliku grafa tijeka podatka i poziva među funkcijama raznih razina detaljnosti.
CFG spada u statički pristup analize izvršnih datoteka te je upotrijebljen kao pristup u mnogim radovima [21] [15] [19] [54] [20] [88]. Cesare u svom radu [21] na taj način predlaže hibridni pristup koji koristi statičku analizu za detekciju pakiranog sadržaja računanjem entropije te dinamičku analizu pri kojoj se otpakirava program izvršavanjem do originalne ulazne točke (engl. *entry point*). U sljedećem koraku se za dobivene kombinacije parova CFG-a računa Levenshteinova udaljenost [21] koja služi za procjenu sličnosti između uzoraka.

Dinamička analiza podrazumijeva analizu ponašanja programa, npr. pokretanjem programa u izoliranoj (zaštićenoj) okolini ili kontroliranjem tijeka izvršavanja pomoću *debuggera*.
Dinamičkom analizom nadziru se:

- Pozivi prema funkcijama operacijskog sustava
- Parametri koji se prosljeđuju funkcijama
- Tijek podataka prilikom izvršavanja programa
- Mrežna aktivnost
- Promjene u registryju i sl.

Analitičari dinamičku analizu najčešće provode u izoliranoj okolini poput virtualnog stroja. Često se u tu svrhu koriste alati Sysinternals - poput ProcessMonitora [79] koji nadzire operacije nad *registryjem*, promjene nad datotekama i mrežnu aktivnost te Process Explorera [78] koji služi za napredno nadziranje aktivnih procesa u operacijskom sustavu. Proces *debugiranja* često se koristi prilikom analize zapakiranih programa. Cilj takve analize je pronaći pravu ulaznu točku programa i promijeniti slijed instrukcija na način da se zaobiđe rutina pakiranja. Najčešće korišteni alati za *debugiranje* su OllyDbg [109] i WinDbg [103]. Za razliku od alata za *debugiranje*, disasembleri, poput najpoznatijeg IdaPro [35], ne pokreću izvršni kôd već se izvršna datoteka pretvara u asemblerske mnemonike. Iako, IdaPro kao najnapredniji alat za analizu izvršnih datoteka ima ugrađen *debugger* i nudi mogućnosti povezivanja s različitim *debuggerima* za sve popularne operacijske sustave (gdb, Bochs emulator, trace replayer).

Metode dinamičke analize možemo podijeliti u sljedeće skupine:

a) *Zaštićene okoline* koriste se za izolirano izvođenje programa, najčešće u virtualiziranoj okolini. U današnje vrijeme popularni su servisi i zaštićene okoline poput Anubisa [39], CWSandboxa [102] i Cuckoo Sandboxa [24] za provođenje dinamičke analize. Takve okoline za razliku od ručne analize cijeli postupak nastoje automatizirati. Većina zaštićenih okolina koristi vritualizacijsku okolinu ili emulator računala, poput QEMU-a [72] ili VirtualBoxa [66].
Početak izvršavanja započinje premještanjem datoteke za analizu s računala udomitelja virtualizacijske okoline u računalo gosta. Ovo se najčešće odvija uz pomoć mehanizma RPC (engl. *remote procedure call*) [11] [24]. Drugi korak je pokretanje datoteke i snimanje njene interakcije s operacijskim sustavom. U tu svrhu se za vrijeme analize danog programa snima mrežni promet pomoću posebnog sučelja i upravljačkog programa (Windows TAP) na mrežnoj kartici. Pozivi prema operacijskom sustavu bilježe se uz pomoć hooking mehanizama [11]. Na ovaj način nadziru

se sistemske funkcije koje se koriste za rukovanje datotekama (npr. NtCreateFile, NtWriteFile, NtReadFile), rukovanje registryjem (npr. NtCreateKey), funkcije za rukovanje procesima i dretvama te funkcije za rukovanje servisima. Zaštićena okolina CWSandbox korištena je za grupiranje malicioznih datoteka u Rieckovom istraživanju [76], dok je Bailey sa suradnicima [10] predstavio klasifikaciju temeljenu na grupiranju poziva prema operacijskom sustavu.

b) *Pozivi prema funkcijama operacijskog sustava* najčešće su dobiveni metodom preusmjeravanja (engl. *hooking*) na korisničkoj razini prema funkcijama Windows API-ja. Pozivi se koriste za stvaranje profila ponašanja izvršnih programa. Za prepoznavanje zlonamjernih programa ponašajni profili programa korišteni su u istraživanjima: Ye i suradnika [106], Shultza [84] i Lanzija [56].

c) *Analiza tragova argumenta* (engl. *taint analysis*) je metoda koja nadzire korištenje i promjene argumenta funkcija i tijek podataka kako bi prepoznala sumnjive pozive i akcije. Analiziranje izvođenja programa moguće je s prikazom grafova tragova [108]. Cilj analize tragova je parcijalna verifikacija same sigurnosti programa. Može se upotrebljavati pri statičkoj analizi [46] i pri analizi izvornog kôda programa pisanih u interpretiranim jezicima. Najčešće se koristi prilikom izvršavanja programa u zaštićenom okruženju [108] [56] [65] gdje se bilježe proslijeđeni argumenti i njihova propagacija kroz pozive prema operacijskom sustavu.

3.2 Ograničenja statičke i dinamičke analize

Statička analiza *Portable Executable* datoteka ima neka ograničenja. Naime, uporabom naprednijih napadačkih metoda poput korištenja tehnika prikrivanja (engl. *obfuscation*) i pakiranja otežava se statička analiza izvršnih datoteka. Korištenjem takvih napadačkih metoda osim otežavanja same detekcije zlonamjernih programa, otežava se ili u potpunosti onemogućuje kvalitetna analiza zlonamjernih programa. Primjerice, usporedimo li dobivene uvezene biblioteke i funkcije dobivene statičkom analizom i snimljene pozive prema operacijskom sustavu dobivene dinamičkom analizom, one se ne trebaju nužno podudarati pri korištenju različitih vrsta analiza. Naime, napadači koriste različite metode kako bi otežali analizu analitičarima i prikrili stvarno ponašanje svog programa. Glavni cilj statičke analize je identifikacija mogućih stanja koje zlonamjerni program može poprimiti, osobito malicioznih stanja koja služe za nanošenje štete žrtvama. Tehnike pakiranja otežavaju proces statičke analize i samu detekciju sigurnosnim alatima. Nužno je zato prethodno otpakirati malver kako bi započeli njegovu analizu. Bez otpakiranog programa statička analiza može ubrzo postati uzaludna.

Postoji nekoliko pristupa, odnosno metoda, koja pokušavaju riješiti problem otpakira-

vanja malicioznih izvršnih datoteka. Najpoznatiji sustavi u trenutačnom stanju znanosti su: Renovo [48], PolyUnpack [77] i OmniUnpack [59].

OmniUnpack je prema rezultatima najnapredniji te je implementiram u obliku upravljačkog programa za Windows i otporan na različite *anti-debug* tehnike. OmniUnpack na početku na sve memorijske stranice postavlja dozvole čitanja i izvršavanja. Prilikom izvršavanja OmniUnpack čeka da procedura otpakiravanja generira iznimku nakon koje će se opet promijeniti dozvole na mogućnost pisanja, ali ne i izvršavanja, te će zapisati nove podatke u memorijske stanice. Time se dobiva otpakirani program zapisan u memoriji koji je potrebno snimiti na tvrdi disk. Poseban modul nadzire opasne pozive prema Windows API-ju te uz pomoć antivirusnog alata iz korisničkog dijela operacijskog sustava prepoznaje maliciozni kôd u novo zapisanim podacima. Pokušaj izvršavanja novo zapisanih podataka uzrokuje iznimku zbog nedostataka dozvole izvršavanja te se pokreće procedura zapisivanja trenutnog (tj. otpakiranog) stanja na tvrdi disk, ova metoda se još naziva i $W \oplus X$ (pisanje ekskluzivno ili izvršavanje) zbog korištenja dozvola nad memorijskim stranicama za generiranje iznimki.

Renovo koristi dinamičku analizu za nadzor promjena tijeka izvršavanja programa pri kojima se nadziranjem memorije prepoznaje skok na novokreirani blok u memoriji koji ujedno predstavlja i ulaznu točku za otpakirani malver. Zadnji korak predstavlja zapisivanje novih stranica iz memorije na tvrdi disk. Glavna Renovova mana je cjelovita emulacija stroja. Temeljen je na virtualizacijskoj platformi TEMU [107], koja je dio okvira BitBlaze, te mu je za analizu pojedinog uzorka programa potrebno do 4 minute, na računalu koji se koristio u njihovim eksperimentima.

Glavni nedostatak dinamičke analize je njezina dugotrajnost, kao što vidimo u slučaju otpakrivanja programa uz pomoć sustava Renovo. Naime, izvršavanje programa u zaštićenoj okolini zahtjeva vrijeme uspostavljanja okoline i vrijeme izvršavanja samog programa. Osim toga napadači koriste napredne tehnike, poput detekcije procesa *debugiranja* i virtualizacijskih okolina, koje dodatno otežavaju provođenje dinamičke analize. Zlonamjerni programi imaju ugrađene mehanizme koje prepoznaju okolinu u kojoj se nalaze i odgađaju izvršenje svoje maliciozne namjere. Tehnike pakiranja isto tako otežavaju dinamičku analizu jer je potrebno pronaći početnu instrukciju (engl. *original entry point*) gdje se maliciozni dio počinje izvršavati i nakon koje možemo donijeti neke zaključke iz snimljenoga slijeda izvršavanja programa.

3.3 Programske značajke korištene u prethodnim istraživanjima

Prema taksonomiji Cesarea i Xianga [22], značajke koje opisuju programe mogu se podijeliti prema svojim karakteristikama u sintaktičke ili semantičke. Sintaktičke značajke predstavljaju tijek i strukturu programa, dok semantičke značajke opisuju ponašanje i izvođenje samog programa. Semantičke značajke dobivaju se dinamičkom analizom izvršnih programa, a sintatičke značajke dobivaju se statičkom analizom programa.
Sintaktičke značajke programa mogu biti: prikaz u jeziku Asembler, prijelazne reprezentacije poput operacijskih kôdova, grafovi tijeka podataka i grafovi poziva funkcija (engl. *call graph*). Dok neke od semantičkih značajki programa predstavljaju: pozivi prema funkcijama operacijskog sustava i tijek podataka pri izvođenju programa.
U operacijskom sustavu Windows postoji nekoliko načina kako presresti pozive prema operacijskom sustavu, odnosno Windows API funkcije. Najjednostavniji način je korištenje biblioteke Microsoft Detours koja preusmjerava API pozive u korisničkom dijelu (ring 3) operacijskog sustava[2]. Drugi način je tehnika preusmjeravanja poziva (engl. *inline hookinga*) koja prepisuje početni bajt-kôd postojeće funkcije programa, čije se izvršavanje preusmjerava *hooking* funkciji zaduženoj za bilježenje poziva. Metodu preusmjeravanja poziva koriste zaštićene okoline poput CWSandboxa [102] i Cuckoo Sandboxa [24].

Za presretanje poziva na razini jezgre operacijskog sustava koristi se upravljački program koji presreće pozive prema modificiranoj SSDT tablici (engl. *System Service Dispatcher Table*). Ovu metodu koriste pojedini *rootkiti* kako bi sakrili svoje akcije. Za Cuckoo Sandbox razvijen je upravljački program zer0m0n[3] koji koristi SSDT *hooking* tehniku.

Koristeći Cesareovu taksonomiju [22] značajke korištene u trenutnačnom stanju znanosti podijeljene su s obzirom na način kojim su dobivene. Njihov pregled prikazan je u tablici 3.1. Iako se rezultati istraživanja uspoređuju po klasifikacijskoj točnosti koriste različite vrste značajki koje se dobivaju različitim vrstama analiza. Zbog toga dan je sažeti pregled na koji načini autori istraživanja su došli do pojednih značajki koje su korištene za klasifikaciju u njihovim metodama.

Strojni kôd je heksadecimalni prikaz programa koji se dobiva iz izvršne datoteke koja sadržava program. Autori najčešće strojni kôd koriste za stvaranje n-grama. Tako autori u istraživanjima [84], [1], [51] traže najčešće n-grame koje opisuju zlonamjerne programe. Kako bi poboljšali klasifikaciju autori često kombiniraju i druge vrste značajki. Najčešći problem kod analize strojnog kôda metodom n-grama je veliki broj generiranih kombi-

[2]Detours, http://research.microsoft.com/en-us/projects/detours/
[3]zer0m0n - https://github.com/conix-security/zer0m0n

Tablica 3.1
Korištene vrste značajki za klasifikaciju malicioznih programa u trenutnom stanju znanosti

Vrsta značajki	Istraživanje
Statička analiza	
Strojni kôd	[84], [1], [52], [96], [7], [95]
Disasemblirani kôd (opcode, asembler mnemonici i sl.)	[96], [82], [40], [7]
Grafovi tijeka podataka	[7], [21], [20]
Strukturne značajke *Portable Executable* datoteke	[106], [69], [96], [7], [87],[81],[74] [83]
Pozivi prema funkcijama operacijskog sustava (uvezene biblioteke)	[84], [106], [69], [96], [3], [40], [87]
Znakovni nizovi koji se nalaze u programu	[84], [96], [40], [87] [83]
Dinamička analiza	
Pozivi prema funkcijama operacijskog sustava	[10],[76], [102], [11], [7], [40], [82] [2] [73] [53]
Instrukcijski (hardverski) tragovi (alat Intel Pin)	[97], [7]

nacija n-grama, iz kojih je potrebno izabrati određeni broj relevantnih n-grama koji će se koristiti u modelu. Tabish i suradnici [95] tako izvršnu datoteku dijele na blokove gdje traže najčešće obrasce strojnog kôda koji koriste zlonamjerni programi, za obrasce izračunavaju više mjera sličnosti (Simpsonov indeks, Canberrin ineks, udaljenost Minkowskog i druge) koje kasnije služe za klasifikaciju i podjelu prema familiji kojoj zlonamjerni program pripada.

Na sličan način autori koriste prikaze disasembliranih izvršnih datoteka poput operacijskih kôdova i asemblerskih mnemonika. U istraživanju Santosa i suradnika [82] iz analiziranih programa izvlače se najčešće sekvence operacijskih kôdova. Islam i suradnici [40] disasembliranu datoteku koriste za stvaranje popisa funkcija i njihovih duljina koje se kasnije koriste kao značajke za klasifikaciju. Od disasembliranih podataka moguće je dobiti CFG graf. CFG u kontekstu klasifikacije i detekcije zlonamjernih prorgama može poslužiti kao izvor značajki. Dobiveni CFG grafovi različitih uzoraka zlonamjernih programa i njihovi dijelovi rangiraju se prema sličnosti. Sličnosti između grafova omogućuju detekciju novih varijacija zlonamjernih programa te njihovu klasifikaciju prema familiji pripadnosti. Cesare [21] u svom istraživanju predlaže detekciju temeljenu na sličnosti između CFG programa, dok Carrera [20] predstavlja praktičnu realizaciju i alate za traženje sličnosti između CFG programa.

Strukturalne značajke izvršne datoteke su sve značajke koje je moguće dobiti analizom formata *Portable Executable*. Nedostatak ovog pristupa je korištenje velikog broja značajki, koji uvode u model mogućnost pojave pretreniranja modela. Santos [81] navodi kako u svom istraživanju koristi 209 značajke dok Shafiq [87] koristi 189 značajke. Autori koji koriste strukturalne značajke često zbog velikog broja značajki ne navode koje značajke koriste i kako ih koriste, primjerice jesu li na njima napravili neku vrstu transformacije. Takav pristup dovodi do nemogućnosti ponavljanja eksperimenta kod drugih istraživača.

Rabaiotti [73] u svojoj doktorskoj disertaciji ukratko navodi strukturalne značajke koje koristi: značajke dobivene iz zaglavlja (*ImageBase*, *SizeOfCode*, *BaseOfCode*, *BaseOfData*, *EntryPoint* i dr.), značajke izvršne sekcije (*VirtualSize*, *VirtualAddress*, *SizeOfRawData*, *PointerToRawData* i dr.) i broj uvezenih funkcija prisutnih u IAT tablici.

Santos i suradnici [81] navode značajke koje koriste u istraživanju te su dodatno primijenili metodu odabira značajki informacijskog prirasta kako bi reducirali početni skup od 209 značajke na suženi skup od 45 značajke. Glavni znanstveni doprinos njihovog istraživanja predstavljaju izvedene značajke te ovdje dajemo njihov skraćeni popis:

- Karakteristike sekcija - broj standardnih sekcija, broj sekcija s veličinom većom od sirovih podataka, kombinacije broj sekcija u odnosu na namjenu (čitanje, pisanje i izvršavanje), najveći i najmanji omjer sirove veličine i veličine pojedine sekcije, broj izvršnih sekcija.
- Entropijske značajke - globalna entropija datoteke, najveća entropija sekcija, srednja vrijednost entropija sekcija, entropija sekcije gdje se nalazi ulazna točka, entropija *code* i *data* sekcije, broj sekcija s graničnim vrijednostima entropije ($7 < Entropija < 8$) i entropija zaglavlja.

Perdisci [69] slično Santosu koristi izvedene značajke za detekciju pakiranih programa. Perdisci koristi sljedeće značajke: broj standarnih i nestandarnih sekcija, broj izvšnih sekcija, broj sekcija namijenjene čitanju ili pisanju ili izvršavanju, broj uvezenih biblioteka koje se koriste za pozive prema operacijskom sustavu te entropiju zaglavlja izvršne sekcije i cijele datoteke.

Analizom *Portable Executable* datoteke često je moguće pronaći u njoj znakovne nizove u čitljivom obliku. Islam i suradnici [40] tako koriste broj jedinstvenih znakovnih nizova kao značajku prilikom klasifikacije programa. Iz tih znakovnih nizova izvlače poznate, po njima opasne, pozive prema operacijskom sustavu. Često autori pozive prema operacijskom sustavu i njihove biblioteke dijele prema njihovoj funkciji. Popularna je podjela na: pozive za rad s datotečnim sustavom, pozive koji služe za rad s *registryjem* i mrežnim

funkcionalnostima ili pozivi koji se koriste za maliciozne namjere poput injektiranja u druge procese [40] [3]. Sustav IMDS [106] pozive prema operacijskom sustavu izvlači iz IAT tablice prilikom analize izvršnih datoteka.

Dinamička analiza obuhvaća pokretanje programa u zaštićenoj okolini. Primjeri dokumentiranih sustava koji provode dinamičku analizu u trenutnom stanju znanosti su CWSandbox [102] i TTAnalyze [11]. Rieck i suradnici [76] tako u svom istraživanju koriste zaštićenu okolinu CWSandbox. Rieck poput istraživanja Baileya i suradnika [10] stvara ponašajne profile od poziva prema operacijskom sustavu koji su snimljeni u zaštićenoj okolini. Ponašajni profili kasnije služe za grupiranje prema familiji zlonamjernog programa. Sličan pristup pri korištenju zaštićene okoline koristi se i u istraživanju Islama i suradnika [40]. U novije vrijeme popularno je snimanje hardverskih indikatora. Od njih se mogu dobiti značajke koje opisuju stanje procesora i njegovih komponenti prilikom izvršavanja programa [7] [97].

3.4 Uspješnost korištenih metoda za klasifikaciju zlonamjernih programa

Strojno učenje posjeduje metode i algoritme pomoću kojih je moguće iz podataka, odnosno prikupljenih značajki programa, izvući određene obrasce (engl. *pattern*) i koristiti ih za klasifikaciju. Prepoznavanje zlonamjernih programa je u suštini klasifikacijski problem s dvije klase - *maliciozni* i *benigni*program. Zlonamjerni programi se mogu grupirati prema vrsti kojoj pripadaju. Takav problem rješava istraživanje Koltera i suradnika [51]. Vrstu zlonamjernog programa određuju njegove funkcionalnosti. Poznajemo sljedeće vrste zlonamjernih programa: trojanski konj, stražnja vrata, *keylogger*, virus, crv i drugi. Drugi autori poput Riecka [76] grupiraju zlonamjerne programe prema familiji kojoj pripadaju. Različiti zlonamjerni programi mogu imati različite varijacije kojima pripadaju npr. familiji Worm.Spybot. Pri označavanju zlonamjernih programa slijedi se nomeklatura gdje se prvo navodi vrsta zlonamjernog programa, njegovo ime i na kraju se može navesti njegova varijacija (npr. Spy.Zbot.YW). Grupiranje zlonamjernih programa prema familiji često služi za izradu filogeneteskog stabla kojim se prikazuje razvoj i povijest varijacija pojedinog zlonamjernog programa [42].

Za potrebe klasifikacije potrebno je imati reprezentativni skup za učenje pomoću kojeg ćemo naučiti parametre modela te skup za testiranje s kojim ćemo odrediti uspješnost modela, odnosno njegove klasifikacije. Istraživanja i korištene metode strojnog učenja možemo podijeliti u odnosu na način kojim su dobivene značajke korištene u klasifikaciji. Neki autori koriste statičku analizu za izvlačenje značajki [52] [51] [84] [106] [69] dok drugi autori koriste dinamičku analizu [76] [108] [10]. Kombinacija statičke i dinamičke analize

je iznimno rijetka, pogotovo na način da se povežu rezultati obje analize.

Rabaiotti u svojoj doktorskoj disertaciji [73] prvi predlaže korištenje statičke i dinamičke analize, međutim nije predstavio rezultate pri korištenju kombiniranih značajki iz obje analize. Autori novijih istraživanja počeli su kombinirati značajke statičke i dinamičke analize pri klasifikaciji. U svojim istraživanjima kombinirane značajke, još nazvane i hibridne, koriste Anderson i suradnici [7], Islam i suradnici [40] i Santos [82].

U trenutnom stanju znanosti autori su problem klasifikacije zlonamjernih programa pokušali riješiti s raznim metodama strojnog učenja. Često isti algoritmi postižu znatno različite rezultate na drugačijim skupovima koji se koriste u istraživanjima.

Shultz i suradnici koriste statičku analizu za izvlačenje značajki, za klasifikaciju zlonamjernih programa koriste jednostavni Bayesov klasifikator i višestruki Bayesov klasifikator te RIPPER metodu za genriranje pravila. Najbolje rezultate u njihovom istraživanju postiže višestruki Bayesov klasifikator koji ima općenitu točnost od 97% uz 6% lažnih pozitiva, dok RIPPER ima općenitu točnost od 89% uz 7% lažnih pozitiva. Autori ne testiraju metode strojnog učenja na istim značajkama. Najbolji rezultat je postigao višestruki Bayesov klasifikator na značajkama dobivenim analizom strojnog kôda.

Sustav MECS Koltera i suradnika [52] [51] za klasifikaciju zlonamjernih programa koristi n-grame strojnog kôda koji se najčešće pojavljuju. Autori su usporedili algoritme stabla odlučivanja C 4.5, metodu potpornih vektora (engl. *support vector machine*), jednostavni Bayes klasifikator, k-najbližih susjeda te su na neke od algoritama primijenili metodu nadopunjavanja slabih klasifikatora (engl. *boosting*). Najbolje rezultate postigla je primjena metode nadopunjavanja na stablo odlučivanja, s površinom ispod krivulje (AUC) od 0.96 uz 5% lažnih pozitiva.

Perdisci i suradnici [69] ne prepoznaju je li program zlonamjeran ili dobronamjeran, već utvrđuju je li program zapakiran. Programi koji su zapakirani često su i zlonamjerni. Perdisci i suradnici uspoređuju više metoda strojnog učenja: jednostavni Bayesov klasifikator, neuronsku mrežu, stablo odlučivanja C 4.5 i metodu k-najbližih susjeda te ansamble pojedinih klasifikatora. Najbolje rezultate postigla je neuronska mreža s općenitom točnošću od 98% i 5% lažnih pozitiva.

Ye i suradnici u svom istraživanju [106] predlažu novu metodu strojnog učenja. Razvili su modificiranu verziju Apriori algoritma nazvanu metoda brzog rasta koja koristi strukturalne značajke. Razvijenu metodu usporedili su s komercijalnim alatima i s drugim algoritmima strojnog učenja (SVM, jednostavni Bayesov klasifikator, stablo odlučivanja).

Njihova metoda je imala najveću općenitu točnost 92% uz 6% lažnih pozitiva.

Rieck i suradnici [76] zlonamjerne programe grupirali su prema familijama. U istraživanju su analizirali izvještaje dinamičke analize koristeći metodu vreće riječi (engl. *bag of words)*, a programe su klasificirali višeklasnim SVM-om. Općenita točnost njihovog pristupa bila je 88%, što je bolje od antivirusnih alata (69%).

Sustav OPEM [82] uspoređuje različite metode strojnog učenja (Bayesove mreže, metodu nasumične šume, metoda k-najbližih susjeda i SVM) na značajkama koje su dobivene statičkom i dinamičkom analizom. Najbolju klasifikaciju imala je metoda potpornih vektora kombiniranjem statičke i dinamičke analize (hibridni pristup) s općenitom točnošću od 95% uz 3% lažnih pozitiva. Prilikom kombiniranja metoda analize eksperimenti se najčešće rade za značajke svake metode analize posebno te za značajke dobivene iz svih korištenih metoda analize.

Najtemeljniji pristup kombiniraju značajki prikazan je u istraživanju i doktorskom radu Andersona [7] [6]. Anderson kombinira šest metoda analize, s time da se kao značajke koriste razni prikazi disasembliranih programa. Primjerice graf tijeka podataka koristi se kao općeniti prikaz kojim se modelira tijek podataka uz pomoć Markovljevog lanaca. Također, Markovljev lanac se koristi za modeliranje rezultata izvođenja u dvije zaštićene okoline. Jedna zaštićena okolina bilježi značajke vezane za rad procesora dok druga prikuplja značajke vezane za komunikaciju promatranog programa s operacijskim sustavom. Anderson kombinira značajke iz svih korištenih metoda analize te za klasifikaciju koristi SVM, za metodu SVM isprobane razne jezgre funkcije. Njegov pristup ima najbolju općenitu točnost od 98% uz 5% lažnih pozitiva.

Islam i suradnici [40] za klasifikaciju zlonamjernih programa koriste popularne algoritme koje koriste i mnogi drugi autori: metodu k-najbližih susjeda, SVM, stablo odlučivanja C 4.5 i metodu nasumične šume. Na metode su primijenili i metodu nadopunjavanja slabih klasifikatora AdaBoost. Važno je napomenuti da su autori testirali poznate vrste zlonamjernih programa, koje su prethodno otpakirali, pošto su posjedovali alate popularne sigurnosne tvrtke. Najbolje rezultate na nezavisnom skupu podataka imala je metoda nasumične šume s primjenom AdaBoost ansambla s ukupnom točnošću od 94% i 8,8% lažnih pozitiva.

Jang je u svojoj doktorskoj disertaciji [42] predstavio sustav BitShred [43]. BitShred je uz Polonium [23] jedan od rijetkih sustava u trenutnom stanju znanosti koji obrađuje ogromnu količinu uzoraka. BitShred je sustav koji omogućuje prepoznavanje sličnih uzoraka zlonamjernih programa. Jangov sustav ne nadmašuje postojeće sustave po točnosti

klasifikacije, već po količini programa koju može dnevno obraditi - u paralelnom načinu rada na Hadoop klasteru omogućuje obradu do 1,9 milijuna uzoraka dnevno. Sustav se sastoji od nekoliko koraka, prvi korak preslikava značajke dobivene n-gram analizom i snimanjem API poziva u sažeti oblik (engl. *feature hashing*). Sažete značajke spremaju se u obliku bit-vektora i nazivaju se BitShread potpisom (engl. *fingerprint*). Sličnost između potpisa dobiva se računanjem Jaccardove udaljenosti između dva potpisa. Sličnost dva uzorka može poprimiti vrijednosti iz intervala $[0, 1]$, odnosno što su uzroci sličniji vrijednost je bliža 1. Dobivena matrica Jaccardovih udaljenosti koristi se za formiranje klastera koristeći hijerarhijsko klasteriranje. BitShred također koristi iskustveni prag t koji uvjetuje broj dobivenih klastera. Za potrebe paralelne obrade prva dva koraka BitShreda prebačena su u Map-Reduce paradigmu (za detaljni postupak pogledati u [42]). BitShred pojedinom zlonamjernom programu dodjeljuje familiju te uzorke zlonamjernih programa prikazuje u obliku filogenetskog stabla koje pokazuje evolucijske odnose između različitih varijacija zlonamjernih programa.

Sustav Valkyrie [53] koristi dinamičke značajke prikupljene telemetrijskim softverom Falcon za zaštitu krajnjih računala kompanije CrowdStrike. Prikupljaju se podaci u prvih 100 sekundi izvođenja svakog programa vezani za procese, mrežni promet i promjene u datotečnom sustavu. Sami telemetrijski podaci povezuju se s podacima koji za analizirani program daje njihova zaštićena okolina. Njihov model može koristiti 868 zbijenih (engl. *dense*) značajki ili 268 milijuna raspršenih (engl. *sparse*) značajki, u istraživanju su usporedili točnosti metode potpornih vektora i nasumične šume za obje vrste značajki. Metoda nasumične šume je s zbijenim značajkama na skupu od 127 000 benignih programa i 78 000 zlonamjernih programa, korištenih u postupku unakrsne validacije s 7 preklapanja, postigla stopu ispravnih pozitiva od 92% uz 5% lažnih pozitiva. Jedan od smjerova budućeg razvoja svoje metode vide u kombiniranju različitih klasifikatora i skupova značajki, kako bi umanjili nedostatke dinamičke analize.

Novija istraživanja, poput onog Saxea i Berlina [83], za klasifikaciju zlonamjernih programa koriste duboke neuronske mreže. Za klasifikaciju se koriste značajke dobivene statičkom analizom, kao što su: entropija dijelova programa, uvezene biblioteke, izvučene vrijednosti te znakovni nizovi iz PE formata. Njihov klasifikacijski model koristi konvolucijsku neuronsku mrežu koja na skupu od oko 430 000 programa prepoznaje zlonamjerne programe s točnošću od 95% uz 0,1% lažnih pozitiva. Također, njihov model je testiran na nepoznatim zlonamjernim programima koji nisu bili sadržani u skupu za učenje, a njegova točnost prepoznavanja zlonamjernih programa iznosila je 67% uz 0,1% lažnih pozitiva. Korištena arhitektura neuronske mreže se sastoji od četiri sloja, ulazni sloj sadrži 1024 značajke izvučene iz PE datoteke, dva sloja s 1024 neurona koriste funkciju aktiviranja ReLU (engl. *rectified linear unit*) te se u zadnjem sloju koristi sigmoidna funkcija aktiviranja. Prva tri

sloja koriste regularizacijsku metodu ispuštanja neurona (engl. *dropout*).

Primjena metoda strojnog učenja i njihovi rezultati uvelike ovise o značajkama koje se koriste i korištenim vrstama analize programa. Na žalost, zbog prirode problema i pojavljivanja novih vrsti zlonamjernih programa ne postoji standardizirani skup koji omogućava ispitivanje i uspoređivanje klasifikatora. Iz tog razloga teško je uspoređivati različita istraživanja. Također, autori u svojim istraživanjima veliki naglasak stavljaju na uspješnost klasifikacije, a ne navode koje obrasce izabrani klasifikator može prepoznati. Veličina skupa za učenje i testiranje često oscilira među istraživanjima, primjerice neka istraživanja koriste skup od 25 zlonamjernih programa i 40 benignih programa [1]. U nekim istraživanjima partneri su sigurnosne tvrtke pa istraživači imaju na raspolaganju i do 100 000 uzoraka zlonamjernih programa [74]. Broj programa s kojime se može provesti istraživanje je oko 1000 uzoraka zlonamjernih i benignih programa, primjerice Anderson [6] koristi 750 zlonamjernih programa i isto toliko benignih programa.

Područje analize zlonamjernih programa je vrlo aktivno te ima nekoliko otvorenih problema koje istraživači nastoje riješiti. Problem od najvećeg interesa je zasigurno prepoznavanje zlonamjernih programa, tj. detekcija nepoznatih varijacija zlonamjernih programa koje potpisi antivirusnih alata ne mogu prepoznati. Od interesa je i proučavanje nasljeđivanja funkcionalnosti i tehnika izrade novih varijacija zlonamjernih programa. Često ovaj postupak ima kao krajnji cilj izradu filogenetskog stabla [42] koje prikazuje odnose između određene populacije promatranih zlonamjernih programa. Prepoznavanje sličnih varijacija i promatranje evolucije takvih uzoraka može pomoći analitičarima prilikom same analize zlonamjernih programa. Sličan problem je i grupiranje sličnih zlonamjernih programa prema njihovoj vrsti. Zlonamjerni programi se grupiraju prema njihovim funkcionalnostima, što omogućuje stvaranje generičkih pravila koji se koriste kao heuristike u sigurnosnim alatima.

Poglavlje 4

Metodologija istraživanja

Cilj istraživanja je razviti novu metodu za prepoznavanje zlonamjernih programa koja koristi značajke statičke i dinamičke analize za klasifikaciju prirode programa. Predložena metoda koristi sustav više nadziranih klasifikatora, uz sljedeća obilježja:

- S obzirom na trenutno stanje u znanosti nova metoda mora biti bolja od trenutačno referentnih znanstvenih istraživanja: Andersona i suradnika [7] [6], Santosa i suradnika [82] te Islama i suradnika [40], što znači da nova metoda mora imati općenitu točnost veću od 90% i stopu lažnih pozitiva manju od 10% za prethodno nepoznate zlonamjerne programe.
- Nova metoda mora biti neovisna o potpisima antivirusnih alata, ali je potpise moguće implementirati u samu metodu.
- Nova metoda ne smije otpakiravati ili disasemblirati programe. Operacija otpakiranja je vremenski zahtjeva za analitičara te ju je vrlo teško generički implementirati zbog postojanja velikog broja slučajeva i vrsta pakiranja, dok je disasembliranje dobronamjernih programa krši uvjete korištenja takvih programa.
- Nova metoda temelji se na kombiniranju rezultata dobivenih iz različitih načina analize programa.
- Nova metoda može se koristiti u postojećim hodogramima analize programa u cilju otkrivanja prethodno nepoznatih zlonamjernih programa.

Stoga nova metoda treba točnije klasificirati zlonamjerne programe od relevantih istraživanja ima mogućnost proširivanja novim klasifikatorima kojima se može dodatno poboljšati klasifikacijska točnost metode. S obzirom na porast broj zlonamjernih programa, nova metoda treba biti u mogućnosti obraditi veliki broj programa i treba imati mogućnost skaliranja na veći broj poslužitelja.
Također, ovo istraživanja ima za zadatak ispitati kako korištenje značajki dinamičke analize, koje nisu dostupne u trenutnoj literaturi, poput argumenata funkcija i vremensko trajanje poziva prema operacijskom sustavu utječu na uspješnost klasifikacije. U odnosu

na referentna istraživanje ovo istraživanje ide korak dalje i pozive prema operacijskom sustavu svrstava u grupe u odnosu na napadačke tehnike koje koriste tvorci malicioznih programa. Grupe napadačkih tehnika, osim mogućnosti označavanje skupa, omogućuju stvaranje modela za prepoznavanje napadačkih tehnika bez korištenja dinamičke analize, tj. korištenjem samo značajki statičke analize. Takvom usporedbom analizom ponašanja programa koju daje dvije različte tehnike obrade programa, tj. statička i dinamička analiza, omogućuje se dolazak do novih znanja u području analize programa i prepoznavanju malicioznog sadržaja.

Hipoteze predloženog istraživanja su sljedeće:

1. Nova metoda ima općenitu točnost klasifikacije bez prethodnog otpakiravanja ili disasembliranja programskog koda, veću od 90% te stopu lažnih pozitiva manju od 10%.

2. Točnost klasifikacije zlonamjernih programa korištenjem hibridnog klasifikatora nadmašuje točnost pojedinačnog klasifikatora koji koristi značajke statičke ili dinamičke analize.

Metodologija istraživanja se sastoji od šest koraka. Koraci obuhvaćaju (1) prikupljanje malicioznih i benignih programa, tj. stvaranje skupova za učenje i i testiranje točnosti klasifikatora, (2) analizu programa i (3) izvlačenje značajki, (4) izbor optimalnog osnovnog klasifikatora (0-reda) za svaki skup značajki, (5) izradu hibridnog klasifikatora te (6) testiranje točnosti novog hibridnog klasifikatora ne neovisnom skupu.

Svaki od koraka korištene metodologije u ovom istraživanju obuhvaća sljedeće:

1. **Prikupljanje skupa programa i izrada stratificranog uzorka malicioznih programa**
   Maliciozni programi prikupljeni su iz otvorenih baza podataka malicioznih programa, dok je skup benignih programa izuzet iz različitih verzija operacijskog sustava Windows te su preuzeti javno dostupni uslužni programi. Nakon prikupljanja dostupnih malicioznih programa u cjeloviti skup koji uključuje i benigne programe, izrađuje se stratificirani uzorak. Stratificrani uzorak koristi se za učenje parametara osnovnih klasifikatora. Stratificirani uzorak malicioznih programa obuhvaća maliciozne programe koji su se pojavili između 1. siječnja 2011. i 1. siječnja 2016.
   Također, prikupljen je i neovisan skup malicoznih programa koji sadrži maliciozne programe koje većina popularnih antivirusnih programa,kada su se pojavili, nije mogla ispravno detektirati. Maliciozni programi iz ovog skupa pojavili su se isključivo poslije 1. siječnja 2016. te su popularni antivirusni alati imali problema prilikom njihove detekcije. Korišteni izvori malicioznih program te sam postupak detaljno je opisan u poglavlju 4.1.

2. **Postupak analize prikupljenih programa**
   Svaki program prolazi kroz dinamičku analizu, tj. izvršava se u zaštićenoj okolini Cuckoo Sandbox te prolazi proces statičke analize. Prikupljeni podaci u ovoj fazi su sirovi podaci koje je potrebno dodatno obraditi i prilagoditi za primjenu u metodama strojnog učenja. Razvijeni okvir koji se koristi za spomenute analize i procesiranje podataka opisan je u poglavlju 4.2.

3. **Ekstrakcija značajki**
   U ovom koraku podaci se prilagođavaju u format pogodan za korištenje u metodama strojnog učenja. Od sirovih prikupljenih podataka iz statičke i dinamičke analize stvaraju se matrice značajki S, D_1 i D_2. Ekstrakcija značajki opisana je u poglavlju 4.2.2.

4. **Izbor metoda strojnog učenja za osnovne klasifikatore**
   Ovaj korak uključuje treniranje i testiranje točnosti odabranih metoda strojnog učenja na stratificranom uzorku malicioznih programa i benignih programa. Za svaki skup značajki odabire se optimalni osnovni klasifikator: C_S, C_{D1} i C_{D2}.

5. **Izrada hibridnog klasifikatora**
   U ovom koraku koristeći odabrane klasifikatore C_S, C_{D1} i C_{D2} izrađuje se novi hibridni klasifikator (1-reda) koji koristi neovisno skupove značajki dobivene statičkom i dinamičkom analizom u osnovnim klasifikatorima (0-reda). Hibridni klasifikator je opisan u poglavlju 4.2.4.

6. **Ispitivanje točnosti hibridnog klasifikatora i usporedba s referentnim pristupima**
   Ovaj korak uključuje usporedbu rezultata hibridnog klasifikatora na neovisnom skupu s antivirusnim alatima i referentnim istraživanjima [7] [40] [82]. Rezultati istraživanja su prikazani u sljedećem poglavlju 5.

4.1 Prikupljanje i izrada skupa programa

Prikupljanje malicioznih programa jedan je od preduvjeta za provođenje ovog istraživanja i kasniju izradu hibridnog klasifikatora. Zbog dinamičnosti područja razvoja malicioznih programa ne postoji standardizirani skup za testiranje novih metoda detekcije malicioznih programa što dodatno otežava izradu novih klasifikatora i njihovu usporedbu s postojećim metodama detekcije. Svi maliciozni programi korišteni u ovom istraživanju prikupljeni su iz repozitorija malicioznih programa: Malshare[1], VirusShare[2] i Malekal Malware Database[3].

Prikupljeni maliciozni programi namijenjeni su daljnjoj obradi, tj. provođenju statičke i dinamičke analize te stvaranja pogodnih značajki iz dobivenih sirovih podatka. U ovom istraživanju korištena su dva skupa malicoznih programa, prvi skup TRAIN-1 sadrži maliciozne programe koji su se pojavili u razdoblju od 1.1.2011. do 1.1.2016., dok drugi skup TEST-1 sadrži novije (aktualne) programe koji su se pojavili poslije nakon 1.1.2016.

Skup TRAIN-1 koristi se učenje parametara i određivanje općenite točnosti postojećih i dobro poznatih algoritama strojnog učenja. Za određivanje općenite točnosti koristi se postupak unakrsne validacije (engl. *cross validation*) s deset preklapanja. Skup TEST-1 sadrži novije maliciozne programe koji su se pojavili u trenutku provođenja opisanih eksperimenata i koji nisu bili poznati većini popularnih antivirusnih alata. Upravo takvo korištenje ažurnih uzoraka malicioznih programa pokazat će uspješnost metode u slučaju stvarnih i novih prijetnji. Za preuzimanje aktualnih malicioznih programa korišten je repozitorij Malekal koje dobar dio antivirusnih alata, uključenih u servis VirusTotal, ne prepoznaje u trenutku njihovog pojavljivanja.

Posebna pažnja posvećena je izradi skupa malicioznih programa namijenjenoga učenju i testiranju algoritama strojnog učenja, tj. odabiru klasifikatora 0-reda. Takav skup treba biti prvenstveno reprezentativan, odnosno mora se izabrati na način da njegova struktura odražava stvarnu zastupljenost pojedinih malicioznih programa. Obzirom na to da godišnja i kvartalna izvješća prijetnji na Internetu često dijele maliciozne sadržaje prema familiji u istraživanju je kao obilježje reprezentativnosti izabrana upravo familija malicioznog programa. Primjerice, većina malicioznih programa iz izvještaja PandaLabsa za treći kvartal 2014.[4] pripada tipu trojanskog konja (njih oko 75%), ostatatak čine virusi (2%), crvi (2%), maliciozni programi koji kradu podatke (7%) i ostali (14%). Za

[1] http://www.malshare.com/
[2] http://virusshare.com/
[3] http://http://malwaredb.malekal.com/
[4] PandaLabs Reports Malware Figures Beat Records with More Than 20 Million New Samples Identified in Q3 2014 - http://www.prweb.com/releases/2014/11/prweb12352269.htm

označavanje familije malicioznog programa korišten servis *VirusTotal*. VirusTotal sadrži najpopularnije antivirusne alate, njih pedesetak, koji se skupno koriste za provjeru prirode analiziranog programa.

Kako oznake familija i imena malicioznih programa variraju između proizvođača antivirusnih alata, u istraživanju se za označavanje familije malicioznog programa isključivo koristi nomenklatura antivirusnog alata Kaspersky. Razlog tomu je njihova konzistencija u imenovanju i dostupnost velikog broja malicioznih programa u njihovoj bazi.

Primjerice Kaspersky označava maliciozni program na sljedeći način
Trojan-Ransom.Win32.Locky.mw.

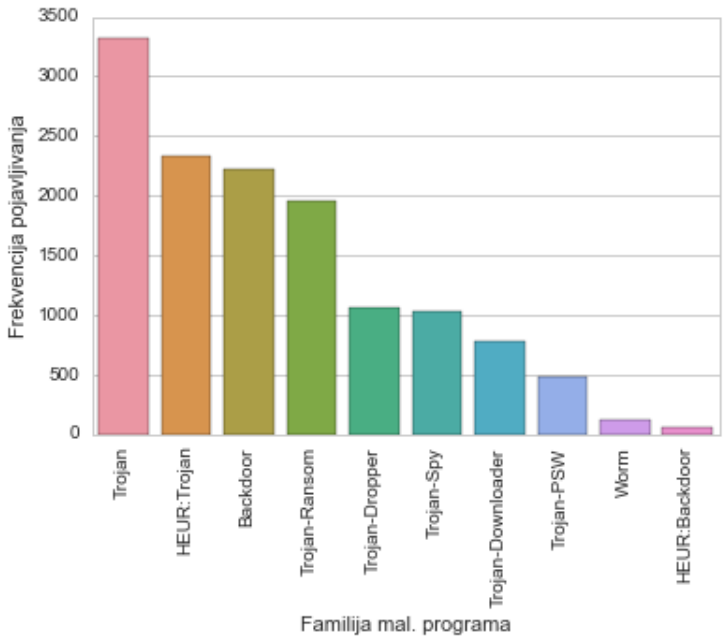
Njihova nomeklatura označavanja ima sljedeće značenje:

- **Trojan-Ransom** označava familiju malicioznog programa. U ovom slučaju ovaj maliciozni program pripada tipu trojanca koji traži otkupninu od svoje žrtve (engl. ransomware).
- **Win32** označava platformu koju cilja maliciozni program, stoga u ovom primjeru maliciozni program cilja 32-bitni operacijski sustav Windows.
- **Locky** je tip malicioznog programa, odnosno njegov naziv. Kako nazive često daju analitičari koji prvi otkriju tip malicioznog programa, a on često održava karakteristiku po kojoj je maliciozni program specifičan ili neki naziv koji je sadržan u malicioznom programu.
- **mw** je varijacija malicioznog programa. Svaki današnji iole popularniji maliciozni program ima na desetke tisuća varijacija. Ova oznaka olakšava samim analitičarima da znaju verziju malicioznog programa ili kojoj kampanji širenja pripada uzorak malicioznog programa.

S repozitorija VirusShare moguće je preuzeti na milijune malicioznih programa, mjerljivo u desetinama terabajta. Ipak, maliciozni programi u sirovom obliku zahtijevaju dodatnu obradu, odnosno slanje na servis VirusTotal kako bi ih se označilo pripadajućom familijom potrebnom za izradu stratificiranog uzorka. Ovo je predstavljalo otežavajuću okolnost jer dnevni limit obrade koje nameće besplatna verzija servisa VirusTotal nije bio dovoljan za obradu većih skupova. Za prikupljanje malicioznih programa i njihovih familija napisane su posebne skripte koje dohvaćaju maliciozne programe i koje prepoznaju familiju malicioznog programa na temelju njegovog opisa u repozitorijima Malekal i Malshare. Samo u iznimnim slučajevima, kada nije postojala klasifikacija alata Kaspersky, koristio se servis VirusTotal za dohvaćanje naziva i familije programa. Također, s VirusSharea ručno su dodani neki uzorci koji su rijetki i svojstveni su za pojedine ciljane napade (engl. *advanced*

persistent threat).

U prvoj fazi prikupljeno je ukupno 32.530 malicioznih programa, koji su se pojavili u razdoblju od 1.9.2011. do 31.12.2015. Za izradu stratificiranog uzorka korišteno je njih 19.877 , iz razloga što odbačeni dio tih programa nije odgovarao opsegu istraživanja, tj. programi nisu bili u formatu *Portable Executable* ili su bili u formatu raznih skripti (npr. JavaScript, Word makroi) kojima su dohvaćali glavni maliciozni programi. Od prikupljenih 19.877 malicioznih programa izuzet je stratificirani uzorak malicioznih programa prema pripadajućoj razdiobi njihovih familija. Konačni stratificirani uzorak sadržavao je 2064 maliciozna programa. Razdioba deset najčešćih familija u stratificiranom uzorku prikazana je na slici 4.1.



Slika 4.1: Razdioba deset najčešćih familija malicioznih programa u stratificiranom uzorku

Algoritam za izradu stratificiranog uzorka malicioznih programa korištenog u skupu TRAIN-1 slijedio je sljedeće korake:

1. Dohvaćanje podataka o svim relevantnim programima u formatu *Portable Executable* s repozitorija Malekal i Malshare.

2. Pretvaranje naziva koji se koristi za označavanja malicioznog programa (Kaspersky format) u pogodan oblik, tj. izvlačenje njegove familije.

3. Izračun relativnih razdioba svake pojedine familije malicioznog programa.

4. Uzimanje slučajno odabranih programa iz cijelog skupa, na temelju relativnih razdioba, pazeći pri tome da se ne ponavljaju isti tipovi malicioznog programa. Prilikom

odabira programa koji se koristi u stratificranom uzorku vodi računa o tipu malicioznog programa koji je odabran te se preskaču isti tipovi programa ukoliko oni postoje.

5. Preuzimanje odabranih malicioznih programa s repozitorija malicioznih programa.

6. Označavanje prirode programa koristeći Cuckoo Sandbox. Nakon analize iz njegovog izvještaja uzima se lista klasifikacija servisa VirusTotal, koja uključuje klasifikacije pedesetak antivirusnih alata. Zatim se spremaju klasifikacije najpopularnijih antivirusnih alata te se na temelju njihovih klasifikacija dodjeljuje oznaka programu. U pravilu ukoliko više od trećine antivirusnih alata prepoznaje program malicioznim dodjeljuje mu se klasa maliciozan, a u suprotnom se označava klasom benigni.

Za konačni skup TRAIN-1, uz maliciozne programe, automatiziranim postupkom prikupljeni su benigni programi koji su dio besplatnih alata operacijskih sustava Windows XP, 7 i 10. Programi su prikupljeni nakon početne instalacije svakog spomenutog operacijskog sustava. Osim toga, korišteni su freeware i shareware uslužni programi koje inače korisnici instaliraju na svojim računalima, kao npr. WinRar, LibreOffice, Gimp i dr., koji su preuzeti sa servisa za distribuciju uslužnih alata poput servisa FileHippo[5]. Skup benignih programa sadrži ukupno 980 programa te je pridodan stratificranom uzorku malicioznih programa. Stratificirani uzorak malicioznih programa i prikupljeni benigni programi čine skup TRAIN-1, koji sadrži ukupno 3044 program, od koji je 32% benigno dok je ostatak 68% maliciozno.

Ze testni skup TEST-1 prikupljeni su programi koji su se pojavili od 1.1.2016. do 31.12.2016., nakon faze filtriranja, tj. izbacivanja duplikata i programa koji nisu kompatibilni s formatom Portable Executable. Izuzeto je 638 malicioznih programa koji su namijenjeni testiranju točnosti novog hibridnog klasifikatora na prethodno nepoznatim uzorcima, koje nisu mogli detektirati ni antivirusni alati. Neovisnom skupu TEST-1 pridodano je 67 benignih programa, izuzetih iz operacijskog sustava Windows 10, koji nisu bili sadržani u skupu za učenje TRAIN-1.

[5] www.filehippo.com

4.2 Postupak analize prikupljenih programa

Prikupljeni skup benignih i zlonamjernih programa TRAIN-1 koriste se u postupku učenja metoda strojnog učenja te će na temelju rezultata biti odabrana adekvatna metoda za svaki skup koja će se koristiti za osnovni klasifikator. Svi prikupljeni programi prolaze kroz postupak statičke i dinamičke analize, a po potrebi se mogu uvesti i varijacije ovih vrsta analiza programa, kao npr. analize drugim zaštićenim okolinama. Podaci dobiveni analizama koriste se za stvaranje značajki potrebnih za učenje klasifikatora te daljnju klasifikaciju korištenjem hibridnog klasifikatora.

Svaki prikupljeni program prolazi kroz proces dinamičke analize, za koju se koristi alat slobodnog koda Cuckoo Sandbox [24]. Cuckoo Sandbox izvršava program u zaštićenoj (virutalnoj) okolini, a pri tome snima interakciju programa s operacijskim sustavom. Posebni modul okoline Cuckoo, nazvan CuckooMon, snima direktne pozive prema operacijskom sustavu, poput: mrežne aktivnosti, promjena u datotečnom sustavu i promjene nad *registryjem*. Cuckoo Sandbox koristi tehnologiju virtualizacije kako bi zaštitio udomiteljsko računalo (engl. *host*) od zaraze zlonamjernim programom i daljnjeg širenja istog kroz mrežu.

Za potrebe istraživanja korišteno je poslužiteljsko računalo sljedećih karakteristika na kojemu se nalazila okolina Cuckoo Sandbox i pripadajući virtualni strojevi za analizu:

- Dva procesora s četiri jezgre Intel Xeon E5405 2.0Ghz
- Radna memorija RAM 20GB
- Tvrdi disk 1TB
- Operacijski sustav Ubuntu 15.04
- Cuckoo Sandox 1.3 s hipoervizorom Virtual Box 5.1

Za potrebe analiza programa korištena okolina Cuckoo Sandbox je dodatno osnažena (engl. *hardening*), iz razloga kako bi se spriječile učestale tehnike koje koriste autori zlonamjernih programa za prepoznavanje zaštićenih okolina i analiziranja programa, poput *anti-vm* i *anti-debug* tehnika.
Nadalje, promijenjene su standardne postavke vezane za virtualizacijski sustav Virtualbox, točnije promijenjen je:

- BIOS virtualnih strojeva koji su se koristili u analizi
- Podaci vezani za tip i serijski broj stroja
- Standardne MAC adrese mrežnih sučelja

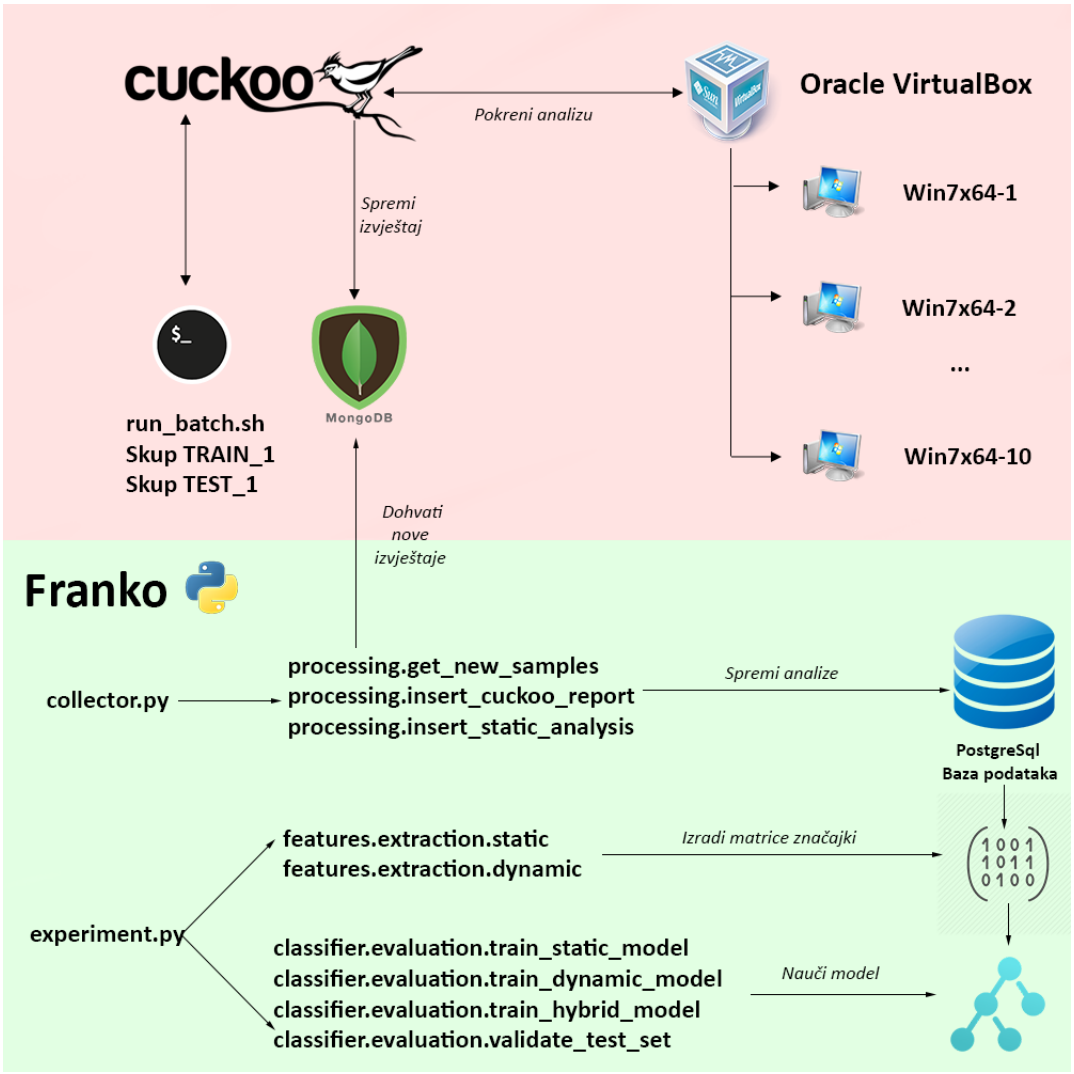
- Svi registry ključevi po kojima se može otkriti da se program pokreće u virtualnoj okolini

Također je promijenjen modul CuckooMon, koji služi za snimanje poziva prema operacijskom sustavu, na način da su dodane lažne povratne poruke koje zavaravaju program u trenutku kada on pokušava prepoznati korištenje virtualne okoline. Kao referenca za osnaživanje zaštićene okoline korištene su javno dostupne upute [71] [67], rad [28] te alat VBoxAntiVMDetect [49]. Za testiranje uspješnosti metoda za detekciju zaštićenih okolina korišten je program PaFish [4].

Jedna od zanimljivijih metoda koju koriste autori zlonamjernih programa za detekciju zaštićenih okolina je korištenje registra TSC (engl. *time stamp counter*), koji služi za mjerenje broja odrađenih ciklusa od strane procesora računala [9]. Ukoliko se oduzmu dvije uzastopne vrijednosti *rdtsc* instrukcija, koja vraća broj odrađenih ciklusa, te se usporedi nekoliko tako izračunatih razlika može se zaključiti da kod fizičkog stroja dobivena razlika je vrlo malena dok kod virtualnog stroja ona varira u velikoj mjeri. Stoga zlonamjerni programi uspoređujući razlike uzastopnih *rdtsc* poziva mogu ustanoviti da se nalaze u virutalnoj okolini te mogu zaustaviti ili odgoditi svoje izvršavanje.

4.2.1 Okvir za analizu zlonamjernih programa Franko

Za potrebe provođenja istraživanja i eksperimenta razvijen je okvir za analizu programa *Franko*. Franko služi za interakciju s okolinom Cuckoo Sandboxom, provođenje statičke analize, ekstrakciju značajki te eksperimente s različitim klasifikatorima strojnog učenja. Za spremanje podatka Franko koristi PostgreSQL bazu podataka. Model podataka detaljnije je opisan u Prilogu A. Franko je napravljen modularno, a na slici 4.2 prikazani su glavni moduli koji se koriste u ovom istraživanju. Modularnost omogućuje proširivanje okvira novim funkcionalnostima te promjenu i podešavanje značajki korištenih klasifikatora, što omogućuje brzo prototipiranje konačnog klasifikatora.
Nakon prikupljanja skupa programa TRAIN-1 i TEST-1 te uspostavljene okoline Cuckoo Sandbox, eksperiment započinje pokretanjem Bash skripte (*run_batch.sh*, slika 4.2) koja služi za slanje programa iz skupa na analizu u Cuckoo Sandbox. Skripta prolazi kroz odabrani skup prikupljenih programa te ih šalje na analizu u Cuckoo Sandbox. Kako bi se spriječilo zagušenje stroja za analizu vodi se računa o maksimalnoj duljini reda čekanja dodijeljenih analiza. Duljina reda jednaka je ukupnom broju dostupnih virtualnih strojeva. Kako završava svaka pojedina analiza, program se miče iz reda čekanja i u red se dodaje novi program čija analiza upravo započinje. Za potrebe istraživanja korišteno je 10 virtualnih strojeva s operacijskim sustavom Windows 7 SP1, svaki virtualni stroj koristi dvije jezgre procesora te 1GB radne memorije.



Slika 4.2: Hodogram aktivnosti okvira za analizu zlonamjernih programa Franko

U prosjeku vrijeme izvršavanja svakog programa u Cuckoo Sandboxu je 200 sekundi, što u provedenim eksperimentima rezultira da se prosjeku svaki sat može analizirati do 150 programa. Maksimalno vrijeme izvršavanja postavljeno je na 300 sekundi, a nakon toga se izvođenje programa prekida i gasi se virtualni stroj. Vrijeme izvršavanja uključuje sve potrebne radnje - paljenje i gašenje virtualnog stroja, učitavanje spremljenog stanja (engl. *snapshot*) sustava te spremanje izvještaja.

Ovaj broj analiza može se poboljšati uvođenjem većeg broja virtualnih strojeva i dodatnih poslužitelja. Povećanje broja dostupnih virtualnih strojeva može se dobiti korištenjem starijeg i po resursima manje zahtjevnog operacijskog sustava Windows XP. Ipak, zbog aktualnosti zlonamjernih programa izabran je operacijski sustav koji prosječni korisnik koristi u svom radu i za koji još uvijek postoje dostupna ažuriranja.

Za svaku analizu bilježimo prvih 10.000 poziva prema operacijskom sustavu kako bi iz-

bjegli mogućnost spremanja velikih količina poziva kad zlonamjerni program prepozna virtualnu okolinu te namjerno izvršava isti poziv u nedogled. Na taj način generiranje izvještaja postaje usko grlo same analize. Kroz istraživanje utvrđeno je da prvih 10.000 poziva predstavljaju sasvim dovoljnu količinu poziva potrebnu za detekciju prirode programa.

Nakon provedene skupne obrade programa iz izabranog skupa, glavni modul sustava Franko (*collector.py*) dohvaća i obrađuje izvještaje iz Cuckoove baze podataka (MongoDB) te sprema obrađene podatke u vlastitu bazu podatka (PostgreSQL). Okvir Franko sprema iz Cuckoove baze podataka prvenstveno podatke vezane za dinamički dio analize, kao primjerice: tijek poziva prema operacijskom sustavu, mrežne konekcije, tragove na tvrdom disku koje program ostavlja i dr. Modul *collector* zadužen je i za provođenje statičke analize. Statičkom analizom pohranjuju se sljedeći podaci o analiziranom programu: uvezene biblioteke (engl. *imports*) ili one koje se daju na korištenje (engl. *exports*) drugim programima, datoteke koje se nalaze u programu, informacije o sekcijama, ukoliko postoji certifikat kojim se programa potpisuje program te ostali općeniti podaci zapisani u zaglavlju PE datoteke. Cuckoo Sandbox posjeduje svoj modul za statičku analizu, ali zbog dodatnih mogućnosti taj modul je dodatno proširen te dodan u okvir Franko.

Postupak analize u okviru Franko zamišljen je da provodi u petlji, tj. izvršni programi se šalju na analizu u Cuckoo Sandbox dok glavni modul čeka da se izvrše analize kako bi spremio potrebne podatke.

4.2.2 Ekstrakcija značajki

Nakon što smo pohranili potrebne podatke iz izvješta Cuckoo Sandboxa te završili statičku analizu, u bazi podataka okvira Franko nalaze se svi potrebni podaci za stvaranje značajki koje ćemo koristiti za klasifikaciju. Iz podatka svake pojedine analize konstruiraju se matrice koje ćemo koristiti za učenje pojedinačnih klasifikatora.

Matrica statičkih značajki S sadrži sljedeće tipove podataka:

- Općeniti podaci o programu dobiveni iz zaglavlja datoteke:
  - Ukupna veličina
  - Datum kompajliranja programa
  - Veličina Import i Export tablica
  - Vrijednosti poravnanja u memoriji i sl.
- Podaci vezane za sekcije u memoriji (virtualne) i one u samoj datoteci programa (engl. *raw*). Ovdje su primjerice izvučene: veličine sekcija, entropija sekcija te

njihovi omjeri virtualne i sirove veličine sekcija koje mogu ukazivati na anomalije u korištenju sekcija. Na primjer, alociranje puno većeg memorijskog prostora nego što sekcija ima u sirovoj veličini može ukazivati na postupak otpakiravanja.

- Tip pakiranja prepoznat na temelju potpisa alata PEID[6].
- Broj uvezenih funkcija po bibliotekama te broj sumnjivih funkcija u odnosu na namjenu (anti-debug, anti-vm, otpakiravnje, kriptiranje i sl.), koje inače koriste zlonamjerni programi. U Prilogu B nalazi se cjelovit popis sumnjivih funkcija koje koriste zlonamjerni programi i koje su uzete u obzir. U obzir je uzeto deset najpopularnijih biblioteka koje koriste zlonamjerni programi, a koje su utvrđene nakon statičke analize na skupu TRAIN-1. Biblioteke koje su uzete u obzir su sljedeće: advapi32.dll, comctl32.dll, gdi32.dll, kernel32.dll, msvcrt.dll, ntdll.dll, ole32.dll, oleaut32.dll, shell32.dll i user32.dll.
- Provjera smještaja zanimljivih karakteristika unutar same PE strukture. Primjerice provjerava se gdje je smještena IAT tablica u PE strukturi, odgovora li smještaj izvršnog kôda ili podataka njegovom smještaju u odgovarajuću sekciji nakon poravnanja u memoriji, nalazi li se ulazna točka programa u izvršnoj sekciji, postoje li neuobičajene vrijednosti baznih adresa i slično.

Pregledom literature u vrijeme pisanja ovog teksta nisu prodađena slična istraživanja koja koriste provjeru adresa unutar *Portable Executable* programa te metode koje provjeravaju pravilnost adresiranja nakon što se program učita u memoriju. Provjera adresa je potaknuta čestim promjenama (engl. *patching*) i prilagođenim pakiranjima zlonamjernih programa. Također, pretpostavka je da kod čestog uređivanja strukture PE formata i prepakiravanja mogu potkrasti greške koje ukazuju na malicioznost datoteke. Primjer ove provjere bio bi testiranje pripada li ulazna točka (engl. *address of entry point*) programa stvarno izvršnoj sekciji nakon učitavanja u memoriju.

Drugi novitet u odnosu na slična istraživanja je korištenje heuristike koje izdvaja biblioteke (engl. *dynamic load library* ili DLL) i njihove funkcije u odnosu na tehnike koje koriste njihovi autori. Na taj se način, primjerice, prepoznaju funkcije pripadajućih biblioteka koje se koriste u detekciji virtualne okoline ili *debuggera*, funkcije koje se koriste za umetanje malicioznog koda u druge pokrenute procese ili funkcije koje se koriste za otpakiravanje pakiranih programa. Za potrebe prepoznavanja takvih funkcija kao osnova korištena je lista sumnjivih funkcija alata *PE Frame* [5] i istraživanje skupine istraživača o učestalim karakteristikama i tehnikama koje otežavaju analizu zlonamjernih programa [17] [16] uključenih u projekt Dissect PE[7]. Matrica S se sastoji od ukupno 94 značajke

[6]PEID - https://www.aldeid.com/wiki/PEiD
[7]Dissect||PE - http://research.dissect.pe/

(stupaca).

Matrica značajki $D1$ sadrži značajke koje održavaju karakteristike poziva koji su izvršeni unutar zaštićene okoline Cuckoo Sandbox. Svakom pozivu prema operacijskom sustavu dodjeljuje se kategorija, kao npr: poziv upravljačkog programa, poziv za operaciju na tvrdom disku, poziv za operaciju u registryju ili poziv za mrežnu aktivnost. Matrica $D1$ sadrži broj pozivanja poziva prema njihovoj kategoriji, statističke podatke o argumentima korištenim unutar poziva i kategorijama kojima pripadaju, uspješnost sustavskih poziva te informacije o vremenskom trajanju i razmaku između poziva. U pregledanim istraživanjama značajke vezne za broj argumenta i vremensko trajanje poziva nisu dostupne te predstavljaju novinu u području detekcije zlonamjernih programa. Matrica $D1$ sadržava 251 značajke.

Matrica $D2$ predstavlja redoslijed poziva koji su izvršeni u zaštićenoj okolini Cuckoo Sandbox. Kako bi se izbjegla velika dimenzionalnost, u obzir je uzeto 50 najučestalijih poziva prema operacijskom sustavu koji su utvrđeni na programima skupa TRAIN-1. Matrica $D2$ za svaki program ima dimenzije 50×50, odnosno može se transformirati u 2500 značajki za svaki analizirani program.

Matrica $D2$ predstavlja vjerojatnost prelaska između uzastopnih poziva i može se predstaviti kao graf poziva $G = (V, E)$ [7], gdje je V čvor koji predstavlja poziv prema operacijskom sustavu, a E težinski brid koji predstavlja tranzicijsku vjerojatnost prema sljedećem (uzastopnom) pozivu. Matrica $D2$ se konstruira na način da se prolaskom kroz listu izvršenih poziva prema operacijskom sustavu vodi računa o zbroju prijelaza između svaka dva uzastopna poziva. Na kraju se broj tranzicija između poziva normalizira dijeljenjem svakog elementa u matrici s ukupnim zbrojem svih prijelaza, tako da svaki redak u matrici ima zbroj od 1.

4.2.3 Odabir metode strojnog učenja za osnovne klasifikatore

Prikupljeni skup izvršnih programa TRAIN-1 koristit ćemo za treniranje standardnih metoda strojnog učenja. Od testiranih metoda za svaki skup značajki odabrat ćemo onu metodu koja postiže najbolji rezultat, tj. ima najveću općenitu točnost, a u slučaju sličnih rezultata odabrat će se metodu kojoj zahtijeva manje vremena za treniranje i klasifikaciju ili su njeni rezultati interpretabilniji. Odabrana metoda strojnog učenja koristi se u za svaki osnovni klasifikator (0-reda) C_S, C_{D1} i C_{D2} te se njihove klasifikacijske vjerojatnosti za svaki analizirani program koristite u hibridnom klasifikatoru.

Za svaku od matrica S, $D1$ i $D2$ evaluirane su metode strojnog učenja koje se uobičajeno koriste u prvim fazama testiranja klasifikacije u određenoj domeni:

- Jednostavni Bayesov klasifikator
- Logistička regresija
- Stablo odlučivanja C 4.5
- Metoda nasumičnih šuma (engl. *Random Forest*)
- Metoda potpornih vektora (engl. Support Vector Machine SVM)

Odabrane metode pripadaju nadziranom učenju pri kojemu je pretpostavka da primjeri u skupu budu označeni pa je tako svaki program u skupu označen s klasom benigni ili maliciozni. Za ispitivanje točnosti algoritama strojnog učenja koristit će se postupak unakrsne validacije s 10 preklapanja (engl. *k-fold cross validation*). Kao što je navedeno u hipotezama, usporedit će se rezultati klasifikacije za značajke dobivene statičkom i dinamičkom analizom s novom hibridnom metodom koja kombinira sve skupove značajki (S, $D1$ i $D2$).

Za svaku metodu strojnog učenja bit će prikazane mjere uspješnosti klasifikacije: stopa pravih i lažnih pozitiva (TPR i FPR), ukupna točnost (engl. accuracy) i ROC-krivulja.

4.2.4 Hibridna metoda

U strojnom učenju postoje metode koje kombiniraju više klasifikatora u cilju donošenja konačne odluke, tj. predikcije. Pretpostavka tih metoda je da će kombinacijom odluka većeg broja eksperata (klasifikatora) rezultat biti točniji nego kada konačnu odluku donosi jedan ekspert. Na taj način metoda ansambla kombinira više istih ili različitih klasifikatora na određenom skupu u cilju poboljšanja točnosti klasifikacije modela u usporedbi s primjenom jedinstvenog klasifikatora na tom istom skupu [91].

Najčešće se ansambli primjenjuju na jednom skupu dijeleći ga u particije te se pojedinačni klasifikatori primjenjuju na svaku particiju zasebno. Dvije popularne metode ansambla koje se primjenjuju jesu: *bagging* i *boosting*. Metoda *bootstrap aggregating* ili *bagging* koristi uzrokovanje s ponavljanjem kako bi od početnog skupa stvorilo različite podskupove na kojima se provodi učenje pojedinog klasifikatora. Predikcije pojedinačnih klasifikatora, naučenih na izuzetim particijama, se u završnoj fazi kod regresijskih problema usrednjavaju, dok se kod klasifikacijskih problema konačna klasa odabire većinskim glasovanjem. Metoda *bagging* se primjenjuje s ciljem smanjenja varijance modela. Za razliku od nje metoda sekvencijalnog jačanja slabih klasifikatora (engl. *boosting*) pokušava točno klasificirati "teške" primjere na kojima klasifikator najčešće griješi, a s ciljem reduciranja pristranost modela. Metoda nasumične šume koristi metodu *bagging*, dok metodu jačanja

slabih klasifikatora (engl. *boosting*) koristi u svom radu metoda AdaBoost [34].

Manje zastupljena metoda iz skupine ansambla, a koja kombinira više klasifikatora u cilju donošenja konačne odluke je Wolpertova metoda slaganja klasifikatora (engl. *stacked generalization*) [105]. Slaganje klasifikatora za konačnu klasifikaciju koristi dvije vrste klasifikatora: osnovne klasifikatore (0-reda) i konačni meta klasifikator (1-reda). Nakon učenja unakrsnom validacijom osnovnih klasifikatora njihove predikcije se koriste kao značajke, odnosno podaci 1-reda, kojima se trenira konačni hibridni klasifikator.

Svojstvo metode ansambla, pa tako i metode slaganja klasifikatora, je da prilikom učenja osnovni skup dijeli na podskupove. Osnovni skup $D = \{(x_n, y_n), n = 1, ..., n\}$ se tako nasumice dijeli na podskupove $D_1, .., Dj$ koji se koriste za učenje klasifikatora 0-reda. Svaki od k klasifikatora uči procesom unakrsne validacije na skupu D te daje modele 0-reda C_k. Za svaku instancu x_n iz skupa D naučeni klasifikator C_k daje predikciju klase z_{kn}, dok y_n predstavlja stvarnu klasu. Na kraju procesa unakrsne validacije dobiveni novi skup sadržava predikcije klasifikatora 0-reda i još se naziva skup 1-reda:

$$D_{CV} = \{(y_n, z_{1n}, .., z_{Kn}), n = 1, ..., N\}$$

Skup 1-reda koristi se za učenje klasifikatora 1-reda H, gdje se također koristi unakrsna validacija. Preporuka Tianga i Wittena [99] je da se umjesto predikcija klasa z_{kn} koriste vjerojatnosti pripadnosti pojedinoj klasi P_{kn}:

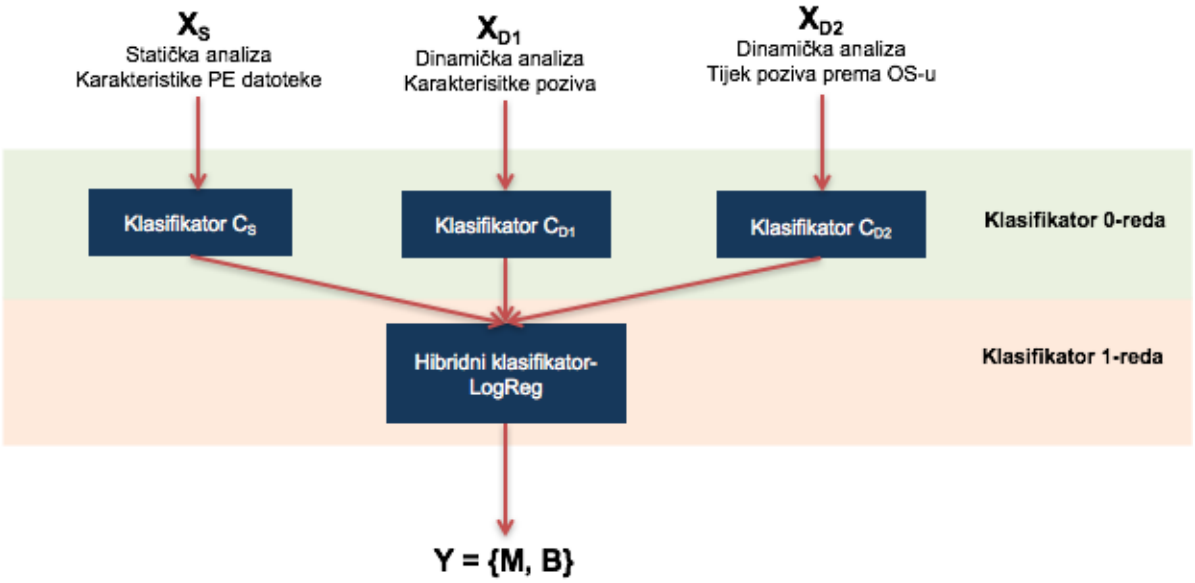
$$D'_{CV} = \{(y_n, P_{1n}, .., P_{Kn}), n = 1, ..., N\}$$

Njihovo istraživanje [99] pokazuje da se korištenjem značajki u klasifikatoru 1-reda u obliku vjerojatnosti pripadnosti pojedinim klasama postižu bolji rezultati u usporedbi s korištenjem predikcijskih klasa. Dva su razloga za ovakvo slaganje različitih klasifikatora, prvi razlog je smanjenje broja grešaka koje se događaju osnovnim klasifikatorima zbog krivo naučenog koncepta dok je drugi povezan s povećanjem generalizacije, tj. točnosti klasifikacije, na novim podacima zbog korištenja većeg broja klasifikatora na različitim podacima.

U ovom istraživanju hibridna metoda koristi varijaciju metode slaganja klasifikatora. Razlika od osnovne Wolpertove definicije je da se ne koristi jedinstveni skup koji se particionira, već različite analize programa daju neovisne skupove za koje se odabire optimalni osnovni klasifikator na temelju najveće općenite točnosti dobivene kroz proces unakrsne validacije. Primjerice, hibridna metoda koristi tri različita skupa S, $D1$ i $D2$. Na temelju usporedbe općenite točnosti metoda strojnog učenja: jednostavnog Bayesovog klasifikatora, logističke regresije, metode potpornih vektora, stabla odlučivanja i metode nasumične šume, odabire

se optimalna metoda za svaki od skupova.

Odabrane metode koriste se kao osnovni klasifikatori C_S, C_{D1} i C_{D2} (slika 4.3), čiji se rezultati, tj. vjerojatnosti, koriste kao značajke u konačnom klasifikatoru. Svaki osnovni klasifikator za klasifikaciju koristi metodu strojnog učenja odabranu na temelju točnosti rezultata odabranih (standardnih) algoritama strojnog učenja. Osnovni klasifikatori daju predikciju u obliku vjerojatnosti pripadnosti pojedinoj klasi (maliciozni ili benigni program) te se predikcije osnovnih klasifikatora koriste kao značajke konačanog hibridnog klasifikatora. isti postupak je korišten za učenje osnovnih klasifikatora i konačnog klasifikatora - unakrsna validacija s 10 preklapanja.



Slika 4.3: Model hibridnog klasifikatora

Broj osnovnih klasifikatora koji ulaze u model trebao bi biti ograničen. Naime, stvaranje dodatnih klasifikatora povećava složenost modela i otežava njegovu konačnu implementaciju. U slučaju uvođenja dodatnog osnovnog klasifikatora potrebno je odabrati pogodnu metodu strojnog učenja koja postiže najbolje rezultate ili one koji daju nove spoznaje vezane za potrebnu predikciju. Primjerice, u slučaju male razlike u općenitoj točnosti možemo se odlučiti odabrati metodu strojnog učenja koja je interpretabilnija za čovjeka ili jednostavno brža.

Poglavlje 5

Rezultati istraživanja

U ovom poglavlju predstavljen je postupak odabira optimalne metode strojnog učenja za svaki od osnovnih klasifikatora te rezultati hibridne metode na skupu za učenje TRAIN-1 i neovisnom skupu TEST-1. Opisane su potrebne transformacije značajki te postupak izbora optimalnih parametara za osnovne klasifikatore. Također, za matrice S i $D1$ dan je pregled rezultata nakon reduciranja skupova značajki i njihove dimenzionalnosti u odnosu na cjeloviti skup značajki. Točnost razvijenog hibridnog klasifikatora uspoređena je s osnovnim klasifikatorima i popularnim antivirusnim alatima. Zaključno je dana procjena vremenskog trajanja pojedinačnih operacija koje sačinjavaju hibridnu metodu.

5.1 Transformacija vrijednosti značajki

Transformacija vrijednosti značajki koristi se kada podaci nisu mjereni na istim skalama, a svrha joj je poboljšanje rezultata pojedinih metoda strojnog učenja ili ubrzanje postupaka učenja parametara same metode. Metode strojnog učenja koje koriste stabla odlučivanja nisu ovisne o razdiobi podataka značajki i za te metode transformacija vrijednosti značajki nije potrebna [75]. Dva najpopularnija načina transformacije značajki jesu normalizacija ili min-max skaliranje te standardizacija ili z-skaliranje.

Normalizacija podataka ili min-max skaliranje omogućava reskaliranje vrijednosti u raspon $[0, 1]$. Za svaku značajku i u pojedinom stupcu matrice koristimo sljedeći izraz kako bi izračunali njenu normaliziranu vrijednost:

$$X_{norm}^{(i)} = \frac{X^{(i)} - X_{min}}{X_{max} - X_{min}}$$

U izrazu X_{min} predstavlja najmanju vrijednost u stupcu matrice, dok X_{max} predstavlja najveću vrijednost neke značajke u stupcu.

Standardizacija podataka primjenjuje se kada želimo prilagoditi vrijednosti normalnoj razdiobi s aritmetičkom sredinom 0 i standardnom devijacijom od 1. Poznato je da algoritmi, poput metode potpornih vektora SVM i logističke regresije, brže nauče težine parametra i postižu bolje rezultate ako podaci koji se koriste odražavaju normalnu razdiobu. Također, za razliku od normalizacije min-max skaliranjem, standardizacija ili z-skaliranje je manje osjetljiva te bolje zadržava stršeće vrijednosti (engl. *outlier*) [75].

Sljedeći izraz primjenjuje se za standardizaciju ili z-skaliranje podataka:

$$X_{std}^{(i)} = \frac{X^{(i)} - \mu_x}{\sigma_x}$$

U njemu je μ_x predstavlja aritmetičku sredina uzorka podataka, dok σ_x predstavlja standardnu devijaciju uzorka podataka.

Na značajkama s velikim rasponima, poput raznih veličina u statičkim značajkama, primijenjene su transformacije min-max i z-skaliranje. Klasifikacijski algoritmi su dali približno jednake rezultate, a iznimka je jednostavni Bayesov klasifikator koji je imao zanemarivo povećanje klasifikacijske točnosti i smanjenje lažnih pozitiva (od oko 1% u oba slučaja).

Transformacijom podataka osim ubrzanja vremena učenja može se koristi za postizanje boljih rezultata klasifikatora. Primjenom standardizacije vrijednosti matrice S metoda logističke regresije postigla je bolje rezultate u usporedbi s korištenjem nestandardiziranih vrijednosti . Nakon standardizacije vrijednosti općenita točnost iznosila je 0,914, a točnost bez normalizacije bila je 0,702. Neki algoritmi poput testiranog SVM-a s radijalnom jezgrenom funkcijom temeljeni su na pretpostavci da su vrijednosti značajki distribuirane oko 0 te da imaju homogene varijance, stoga se transformacijom postiže veća točnost klasifikacije i ubrzava proces učenja parametara metode.

5.2 Odabir optimalnih parametara za osnovne klasifikatore

Matrica značajki statičke analize S i matrice dinamičke analize $D1$ i $D2$ korištene su za treniranje i odabir metoda strojnog učenja koje će se koristiti u osnovnim klasifikatorima C_S, C_{D1} i C_{D2}. Kao što je prethodno opisano, skup TRAIN-1 korišten je za podešavanje parametara metoda strojnog učenja te za odabir opitmalnog klasifikatora za svaki od skupova.

Kako svaka metoda strojnog učenja ima podesive parametre o kojima ovisi točnost njihove klasifikacije, na svakom skupu značajki S, $D1$ i $D2$ isprobani su različiti parametri s ciljem odabira optimalnih parametara, tj. kombinacije parametara, s kojima se postiže najveća točnost. Za izbor parametara uzeta je u obzir mjera općenite točnosti (engl. *accuracy*) dobivena nakon testiranja mogućih kombinacija parametara u postupku unakrsne validacije s 10 preklapanja. Za traženje optimalnih parametara korišten je algoritam iscrpnog traženja (engl. *grid search*) [26] koji u obzir uzima sve kombinacije parametara koje želimo podesiti. Kombinacije parametra koje su degradirale brzinu izvođenja prilikom eksperimenta eliminirani smo iz daljnjeg izbora.

Rasponi parametara algoritama strojnog učenja odabrani su prema principu dobre prakse te su vrijednosti oko izabranih parametara još dodatno provjereni u dodatnom eksperimentu, a za svaki osnovni klasifikator isprobani su sljedeći parametri:

- Kod logističke regresije isprobane su vrijednosti parametra C: 0,0001, 0,001, 0,01, 0,1, 1, 10, 100, 1000 te vrste regularizacije L1 i L2. Parametar C se odnosi na regularizaciju, njegove manje vrijednosti upućuju na jaču regularizaciju.
- Za jednostavni Bayesov klasifikator izabran je parametar α, koji se koristi za postupak aditivnog zaglađivanja, a testirane su vrijednosti za α: 0, 0,01 , 0,5 , 1, 10, 100, 100. Parametar α utječe na izbacivanje nultih vjerojatnosti pripadnosti značajki pojedinoj klasi.

- Kod metode potpornih vektora isprobane su različite jezgrene funkcije: linearna funkcija, radijalna jezgra i polinomna jezgra drugog i trećeg stupnja. Testiran je parametar regularizacije C u rasponu vrijednosti: 0,1 , 0,5 , 1, 10, 100 te parametar γ koji se koristi kod radijalne jezgre s vrijednostima: 0,001 , 0,1 , 0,5 , 1 i 10. Parametar C koristi se za regularizaciju, tj. za definiranje regije razdvajanja. Male vrijednosti parametra C omogućuju glađe regije razdvajanja odnosno veći razmak između margine razdvajanja, a većim vrijednostima parametra C odabire se hiperravnina s manjom marginom razdvajanja pri čemu se nastoji točno klasificirati što veći broj primjera. Parametar γ predstavlja korektivni faktor koji služi za pridavanje važnosti pojedinim primjerima u konačnoj klasifikaciji. Malena vrijednost γ obuhvaća širok prostor oko promatranog primjera, dok velika γ predstavlja manji prostor.

- Kod stabla odlučivanja testirani su sljedeći parametri: mjere kvalitete razdvajanja (Gini indeks i Informacijski dobitak), najmanji broj primjera potrebnih za razdvajanje čvora u stablu (2, 10, 20), maksimalna dubina stabla (0, 1, 2, 5, 10) te najmanji broj primjera potrebnih za krajnji čvor (list) u stablu (1, 5, 10).

- Za metodu nasumične šume testirani su sljedeći parametri: broj korištenih stabla odlučivanju u ansamblu (1, 5, 10, 20, 30, 50, 100), dubina svakog stabla (neograničena, 1, 2, 5, 10) i broj značajki korištenih za traženje optimalnog razdvajanja ($log(n)$, $\sqrt{n}$)

U tablicama 5.1, 5.2 i 5.3 dan je pregled pronađenih optimalnih parametara za svaki od osnovnih klasifikatora na skupovima značajki S, $D1$ i $D2$. Za vrijednosti značajki korištenih kod stabla odlučivanja i metode nasumične šume nije primijenjena nikakva transformacija, dok su za logističku regresiju vrijednosti transformirane z-skaliranjem, a kod metode potpornih vektora SVM primijenjeno je z-skaliranje vrijednosti matrice S i $D1$.

Tablica 5.1
Odabrani parametri s navjećom općenitom točnošću za matricu S

Metoda s parametrima	Općenita točnost
Logistička regresija (L2 regularizacija, C=0,1, uz z-skaliranje)	0,9137
Jednostavni Bayesov klasifikator (α=0,01)	0,8484
SVM (Linearna jezgrena funkcija, C=100, korišteno z-skaliranje)	0,9144
SVM (RBF, C=10, γ=0,5 uz z-skaliranje)	0,9462
SVM (Polinomna jezgra funkcija stupanj=3, C=0,1, γ=1, korišteno z-skaliranje)	0,9442
Stablo odlučivanja (Mjera razdvajanja Gini indeks, broj primjera potrebnih za razdvajanje čvora 20, broj primjera potrebnih za list 5, dubina stabla 10)	0,9408
Metoda nasumičnih šuma (Broj značajki $\sqrt{n}$, Broj stabla 100, neograničena dubina pojedinačnog stabla)	0,9611

Za matricu S, logistička regresija postiže dobre rezultate za male vrijednosti parametra C, odnosno koristeći veću regularizaciju. Najbolji rezultat postiže koristeći L2 regularizaciju uz C=0,1, s općenitom točnošću od 0,9137. Jednostavni Bayesov klasifikator za α=0, tj. kad se ne koristi zaglađivanje, postiže loše rezultate (općenita točnost 0,313). Najbolji rezultat postiže se za α=0,01, čime se sprječava da pojedine značajke imaju nulte vjerojatnosti pripadnosti pojedinoj klasi. Metoda potpornih vektora SVM najbolju općenitu točnost od 0,9462 postiže koristeći radijalnu jezgrenu funkciju uz malu regularizaciju C=10 i γ=0,5, a prethodni szu vrijednosti značajki standardizirane. Stablo odlučivanja najbolju općenitu točnost od 0,9408 postiže koristeći mjeru razdvajanja Gini indeksa, najmanje 20 primjera je potrebno za razdvajanje čvora u stablu, najmanje 5 primjera je potrebno za stvaranje lista te uz maksimalnu dubinu stabla od 10 razina. Kod stabla odlučivanja točnost je povezana s dubinom stabla, tj. manja dubina stabla uzrokuje manju točnost klasifikacije. Metoda nasumične šume najbolji rezultat postiže za $\sqrt{n}$ značajki, neograničenu dubinu za svako pojedino stablo i broj stabla 100, što daje općenitu točnost 0,9611.

Tablica 5.2
Odabrani parametri s navjećom općenitom točnošću za matricu $D1$

Metoda s parametrima	Općenita točnost
Logistička regresija (L2 regularizacija, C=1, uz z-skaliranje)	0,9313
Jednostavni Bayesov klasifikator (α=100)	0,8058
SVM (Linearna jezgrena funkcija, $C = 100$, uz z-skaliranje)	0,9289
SVM (RBF, C=100, γ=0,1 uz z-skaliranje)	0,9469
SVM (Polinomna jezgra funkcija trećeg stupanja, C=10, γ=0,1 uz z-skaliranje)	0,9486
Stablo odlučivanja (Mjera razdvajanja Gini indeks, najmanji broj primjera potrebnih za razdvajanje čvora u stablu 2, najmanji broj primjera potrebnih za stvaranje lista 1, dubina stabla 10)	0,9208
Metoda nasumičnih šuma (Broj značajki $\sqrt{n}$), Broj stabla 50, neograničena dubina pojedinačnog stabla)	0,9512

Za matricu $D1$, logistička regresija postiže najbolju općenitu točnost od 0,9313 za vrijednost parametra C=1, veća regularizacija (manja vrijednost parametra C) u pravilu nosi lošije rezultate. Jednostavni Bayesov klasifikator za manje vrijednosti parmetra α postiže lošije rezultate, a najbolju općenitu točnost 0,8058 postiže za $\alpha = 100$. Metoda potpornih vektora SVM najbolju općenitu točnost 0,9486 postiže koristeći polinomnu jezgrenu funkciju trećeg stupnja i C=10. Stablo odlučivanja za dubinu stabla 10, najmanji 2 primjera potrebnih za razdvanje čvora i najmanji jedan primjer potreban za stvaranje lista postiže općenitu točnost od 0,9208, kao i kod matrice S manja dubina stabla uzrokuje manju točnost klasifikacije - za dubinu s jednom i dvije razine općenita točnost je 0,81. Metoda nasumične šume korištenjem $\sqrt{n}$ značajki, uz neograničenu dubinu koju može postići svako pojedino stablo te korištenjem 100 stabla odlučivanja u ansamblu postiže najveću općenitu točnost od 0,9512. Kao i kod stabla odlučivanja manja dubina pojedinog stabla utječe na smanjenje točnosti.

Tablica 5.3
Odabrani parametri s navjećom općenitom točnošću za matricu $D2$

Metoda s parametrima	Općenita točnost
Logistička regresija (L2 regularizacija, C=1000, uz z-skaliranje)	0,8826
Jednostavni Bayesov klasifikator (α=100)	0,7875
SVM (RBF, C=100, γ=10)	0,9272
Stablo odlučivanja (Mjera razdvajanja Gini indeks, Broj primjera potrebnih za razdvajanje čvora 10, broj primjeri potrebni za list 5, neograničena dubina stabla)	0,8799
Metoda nasumičnih šuma (Broj značajki $\sqrt{n}$), Broj stabla 50, neograničena dubina pojedinačnog stabla)	0,9157

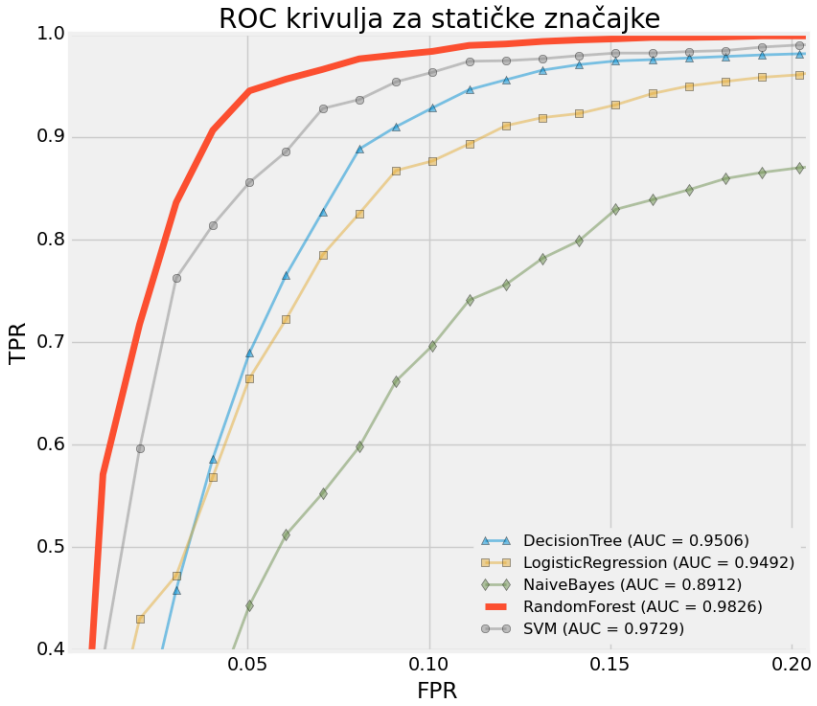
Na matricu $D2$ primijenjena je dekompozicija na glavne komponente (PCA) gdje se matrica redoslijeda sistemskih poziva svakog primjera dimenzija 50×50 svodi s 2500 značajki na 50 glavnih komponenata koje se koriste za klasifikaciju. Logistička regresija najbolju općenitu točnost od 0,8826 postiže većom vrijednosti parametra C=1000 što pokazuje da nije potrebna dodatna regularizacija. Primjena manjeg parametra C rezultirala je padom općenite točnosti klasifikacije. Promjena parametra α kod jednostavnog Bayesovog klasifikatora nije znatno utjecala na općenitu točnost. Najbolja općenita točnost od 0,7875 postignuta je za parametar α=100. Metoda SVM je imala najbolju općenitu točnost 0,9272 koristeći radijalnu jezgrenu funkciju, s parametrima C=100 te γ=10. Nešto slabiju općenitu točnost 0,9167 metoda SVM je postigla korištenjem polinomne jezgrene funkcije trećeg stupnja. Stablo odlučivanja postiže najbolju općenitu točnost 0,8799 kada nema ograničene dubine stabla, za stvaranje novog čvora potrebno je najmanje 10 primjera te je potrebno najmanje 5 primjera za stvaranje svakog lista u stablu. Metoda nasumične šume najbolju općenitu točnost 0,9157 postigla je bez ograničenja dubine pojedinačnog stabla u modelu, koristeći 50 stabla odlučivanja u ansamblu te $\sqrt{n}$ značajki korištenih za traženje optimalnog razdvajanja.

5.3 Odabir osnovnih klasifikatora C_S, C_{D1} i C_{D2}

Nakon traženja optimalnih kombinacija parametara svake metode strojnog učenja na danom skupu značajki, odnosno matricama S, $D1$ i $D2$, potrebno je za svaki od skupova odabrati optimalni osnovni klasifikator (0-reda) čiji će rezultati predstavljati značajke 1-reda potrebne za konačnu klasifikaciju u hibridnoj metodi (meta ili klasifikator 1-reda). Kao što je već spomenuto za procjenu točnosti korištena je metoda unakrsne validacije s 10 preklapanja, a smanjenje broja preklapanja na 5 preklapanja nije znatno utjecalo na rezultate pa se u konačnom eksperimentu koristi 10 preklapanja.

U tablici 5.4 prikazani su rezultati klasifikatora za svaku od matrica.

Za skup S svi klasifikatori postižu općenitu točnost veću od 90% osim jednostavnog Bayesovog klasifikatora (84,5%). Najmanju stopu lažnih pozitiva (FPR) ima metoda nasumične šume, njih 9,7%. Stopa detekcije zlonamjernih programa (istinitih pozitiva, TPR) je približno jednaka za klasifikatore: stabla odlučivanja, metodu potpornih vektora SVM i metodu nasumične šume. Metoda nasumične šume ima najveću stopu istinitih pozitiva 98,57% i općenitu točnost od 95,59%.



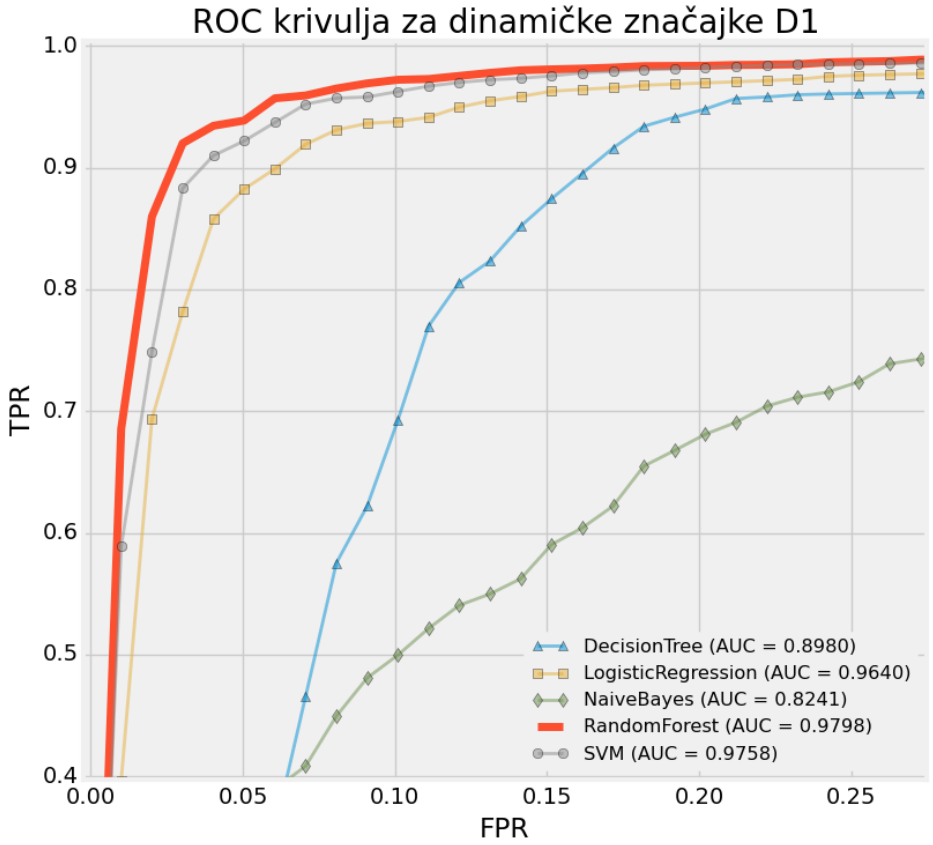
Slika 5.1: Usporedba ROC krivulja testiranih metoda strojnog učenja u svrhu odabira optimalnog osnovnog klasifikatora za statičke značajke (matrica S)

Tablica 5.4
Usporedba rezultata testiranih metoda strojnog učenja za matrice S, $D1$ i $D2$ u svrhu odabira optimalnog osnovnog klasifikatora

Skup	Klasifikator	TPR	FPR	TNR	FNR	Općenita točnost
S	Logistička regresija	0.954657	0.184466	0.815534	0.045343	0.911028
S	Jednostavni Bayesov klasifikator	0.855101	0.175836	0.824164	0.144899	0.845399
S	SVM	0.974372	0.111111	0.888889	0.025628	0.947564
S	Stablo odlučivanja C 4.5	0.961557	0.112190	0.887810	0.038443	0.938430
S	**Metoda nasumične šume**	**0.985707**	**0.097087**	0.902913	0.014293	**0.959743**
D1	Logistička regresija	0.947265	0.106796	0.893204	0.052735	0.930311
D1	Jednostavni Bayesov klasifikator	0.936422	0.472492	0.527508	0.063578	0.808187
D1	SVM	0.954165	**0.077670**	0.922330	0.045835	0.944181
D1	Stablo odlučivanja C 4.5	0.941350	0.143474	0.856526	0.058650	0.914750
D1	**Metoda nasumične šume**	**0.971907**	0.096009	0.903991	0.028093	**0.950609**
D2	Logistička regresija	0.944800	0.185545	0.814455	0.055200	0.903924
D2	Jednostavni Bayesov klasifikator	0.938393	0.553398	0.446602	0.061607	0.784168
D2	**SVM**	0.947265	**0.105717**	0.894283	0.052735	**0.930650**
D2	Stablo odlučivanja C 4.5	0.912765	0.184466	0.815534	0.087235	0.882273
D2	Metoda nasumične šume	**0.952686**	0.154261	0.845739	0.047314	0.919147

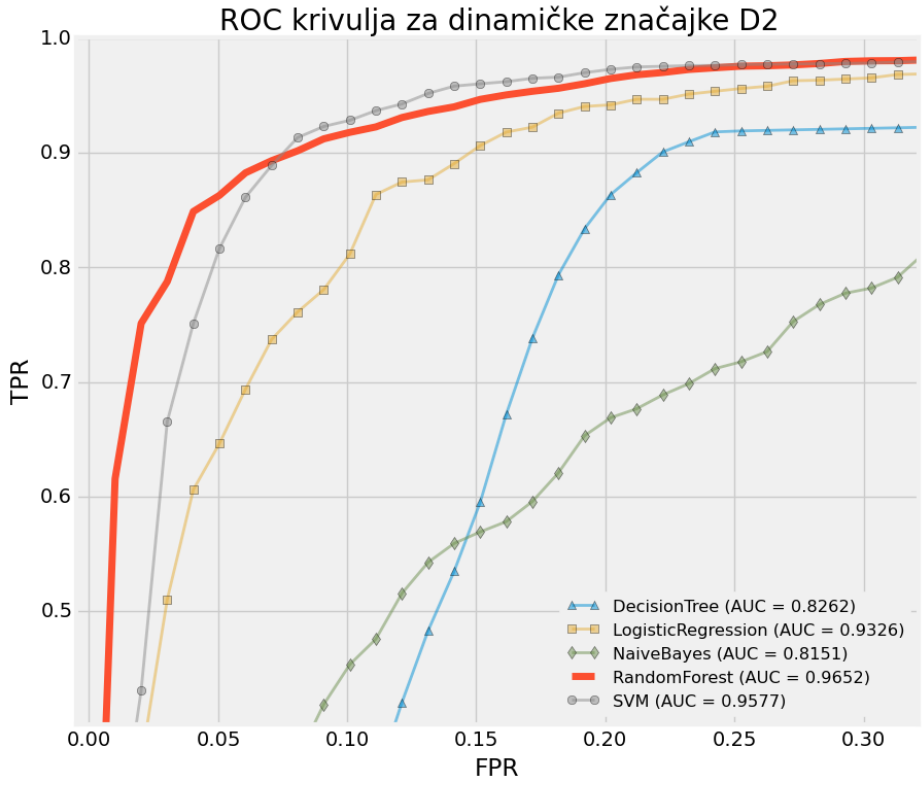
Na slici 5.1 prikazane su usporedne ROC krivulje klasifikatora koji koriste skup statičkih značajki S. Najbolje rezultate postigla je metoda nasumične šume s površinom ispod krivulje AUC 0,9826. Na temelju postignutih rezultata i niske stope lažnih pozitiva metoda nasumične šume koristit će se kao klasifikator C_S u hibridnom klasifikatoru.

Za skup $D1$ najveću općenitu točnost (95,06%) postiže metoda nasumične šume. Metoda potpornih vektora SVM ima najmanji broj lažnih pozitiva (7,7%) u odnosu na ostale metode, za usporedbu metoda nasumične šume ima neznatno veći postotak lažnih pozitiva 9,6%. Usporedne ROC krivulje za testirane osnovne klasifikatore na matrici dinamičkih značajke $D1$ prikazane su na slici 5.2, a najbolji rezultat na skupu TRAIN-1 postigla je metoda nasumične šume s površinom ispod krivulje 0,9798. Metoda potpornih vektora SVM postigla je slične rezultate kao i metoda nasumične šume, njena površina ispod krivulje iznosila je 0,9758. Vrlo dobre rezultate na matrici D_1 postigla je i logistička regresija (AUC=0,9640). Za matricu $D1$ dakle postoji više metoda s približno istim točnostima, ipak zbog boljih rezultata u hibridnoj metodi kao klasifikator C_{D1} koristit će se metoda nasumične šume.



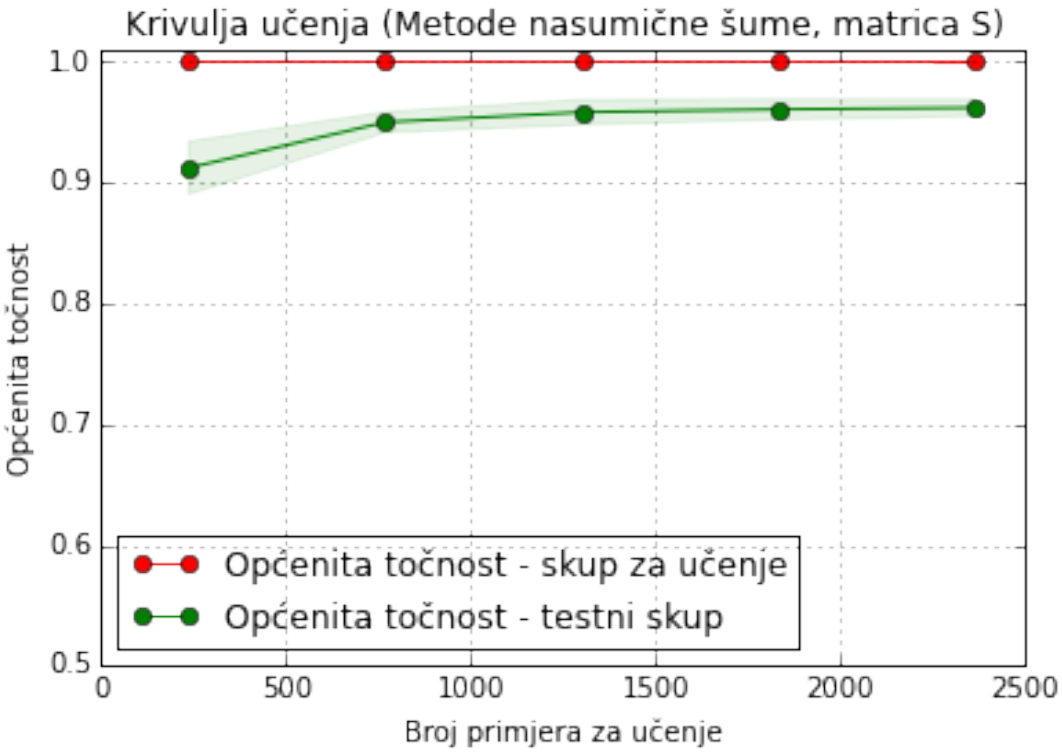
Slika 5.2: Usporedba ROC krivulja testiranih metoda strojnog učenja u svrhu odabira optimalnog osnovnog klasifikatora za dinamičke značajke (matrica $D1$)

Slično kao i kod skupa $D1$ na skupu $D2$ najbolje rezultate postigle su sljedeće metode: logistička regresija, metoda nasumične šume i metoda potpornih vektora SVM. Najbolju općenitu točnost postigla je metoda potpornih vektora 93,06%, dok logistička regresija je imala općenitu točnost od 90,39%, a metoda nasumične šume općenitu točnost od 91,91%. Najmanju stopu lažnih pozitiva imala je metoda potpornih vektora SVM 10,57% dok je metoda nasumične šume imala stopu lažnih pozitiva 15,42%. Usporedbom ROC krivulja testiranih klasifikatora prvog reda vidljivo je da najveću površinu ispod krivulje ima metoda nasumične šume (AUC=0,9652). Kao klasifikator C_{D2} u hibridnom klasifikatoru koristit će se metoda nasumične šume zbog korištenja jedinstvene metode strojnog učenja za sve osnovne klasifikatore.

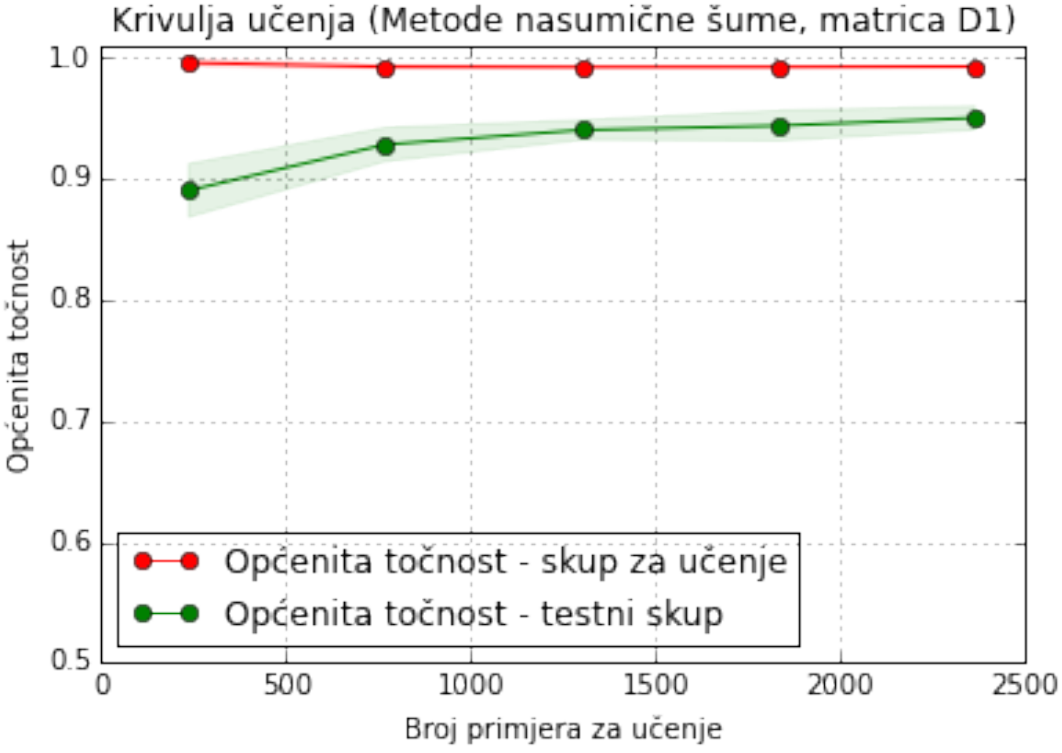


Slika 5.3: Usporedba ROC krivulja testiranih metoda strojnog učenja u svrhu odabira optimalnog osnovnog klasifikatora za značajke (matrica $D2$)

Na slikama 5.4 i 5.5 prikazane su krivulje učenja za odabranu metodu nasumične šume prilikom korištenja svih značajki S i $D1$ na skupu za učenje TRAIN-1. Iz krivulja je vidljivo da metoda nasumične šume, koja je inače sklona pojavi pretreniranosti, na matricima S i $D1$ postiže dobre rezultate te nema značajnije razlike između općenite točnosti na skupu za učenje i izuzetom testnom skupu. Može se zaključiti da je broj programa korišten u skupu za učenje dovoljan te da izabrana metoda strojnog učenja ne pokazuje značajnije pojave podtreniranosti (engl. *under-fitting*) ili pretreniranosti (engl. *over-fitting*).



Slika 5.4: Krivulja učenja za matricu S

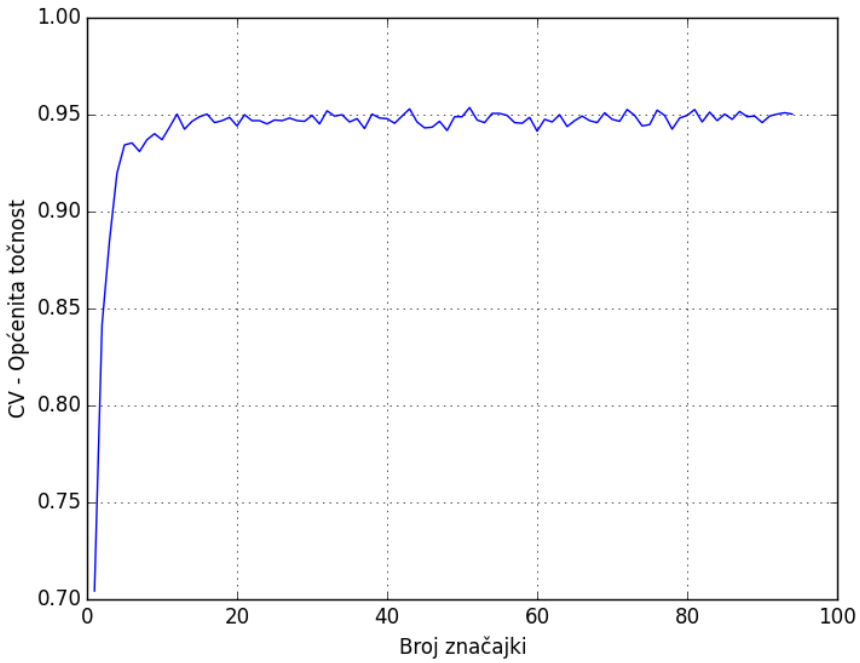


Slika 5.5: Krivulja učenja za matricu $D1$

5.4 Odabir relevantnih značajki i redukcija dimenzionalnosti skupova

Veliki broj značajki, koje je moguće dobiti iz analize programa, može loše utjecati na točnost modela prilikom klasifikacije novih uzoraka programa. Umjesto da naučeni model dobro generalizira pri klasifikaciji, on je prilikom učenja naučio šum, odnosno veliki raspon varijance, što uzrokuje loše rezultate na testnom skupu. Kako bi smanjili pojavu pretreniranja (engl. *overfitting*) i pojednostavili same modele, na matrice S i $D1$ su primijenjene metode odabira značajki i redukcije dimenzionalnosti. Korištene su metode rekurzivne eliminacije (RFE) i metoda glavnih komponenata (PCA) te je prikazana usporedba rezultata klasifikacije reduciranog skupa značajki s cjelovitim skupom značajki.

Za matricu $D2$ nisu prikazani usporedni rezultati jer zbog njene velike dimenzionalnosti, tj. svaki primjer je bio opisan matricom poziva prema operacijskom sustavu dimenzija 50×50, bilo potrebno za početak primijeniti redukciju dimenzionalnosti kako bi se spriječile oscilacije u rezultatima i dugo trajanje procesa učenja. U tu svrhu primijenjena je metoda glavnih komponenta PCA gdje je svaki primjer s početnih 2500 značajki sveden na 50 glavnih komponenata koje su korištene za klasifikaciju u osnovnom klasifikatoru C_{D2}. Za matricu $D2$ odabranih 50 glavnih komponenata zadržava 95% varijance cjelovitog skupa značajki.



Slika 5.6: Utjecaj redukcije broja značajki metodom rekurzivne eliminacije na općenitu točnost (matrica S). Model s 51 značajke postiže općenitu točnost 0.9536

Prilikom korištenja metode rekurzivne eliminacije (RFE) za svaki podskup značajki korištena je metoda nasumične šume i unakrsna validacija s deset preklapanja za određivanje općenite točnosti. Matrica S sadrži ukupno 95 značajke koje je moguće značajno reducirati metodom RFE na 51 značajke uz prihvatljivu točnost klasifikacije. U eksperimentu je model s 51 značajkom postigao najveću općenitu točnost od 96,17%. Na slici 5.6 može se primijetiti da 10 najbolje rangiranih značajki daju općenitu točnost, koja je približno jednaka najvećoj točnosti postignutoj prilikom eksperimenta, koristeći 51 značajku. U tablici 5.5 prikazana je usporedba rezultata klasifikatora koji koriste cjeloviti skup značajki i reducirane skupove značajki metodama RFE i PCA. Iz rezultata je moguće primijetiti da primjenom logističke regresije i metode potpornih vektora SVM-a na reducirani skup značajki, metodom RFE (n=51), klasifikatori postižu lošije rezultate od slučaja kada se koristi cjeloviti skup značajki. Lošiji rezultati se očituju u povećanju stope lažnih pozitiva, osim u slučaju kada se koristi metoda nasumične šume koja postiže slične rezultate kao i kod korištenja cjelovitog skupa značajki, a općenita točnost u oba slučaja je 96%. Dakle, prema rezultatima u hibridnom klasifikatoru se za klasifikator C_S može koristiti metoda nasumične šume s podskupom od 51 najznačajnije značajke izabrane metodom rekurzivne eliminacije (RFE).

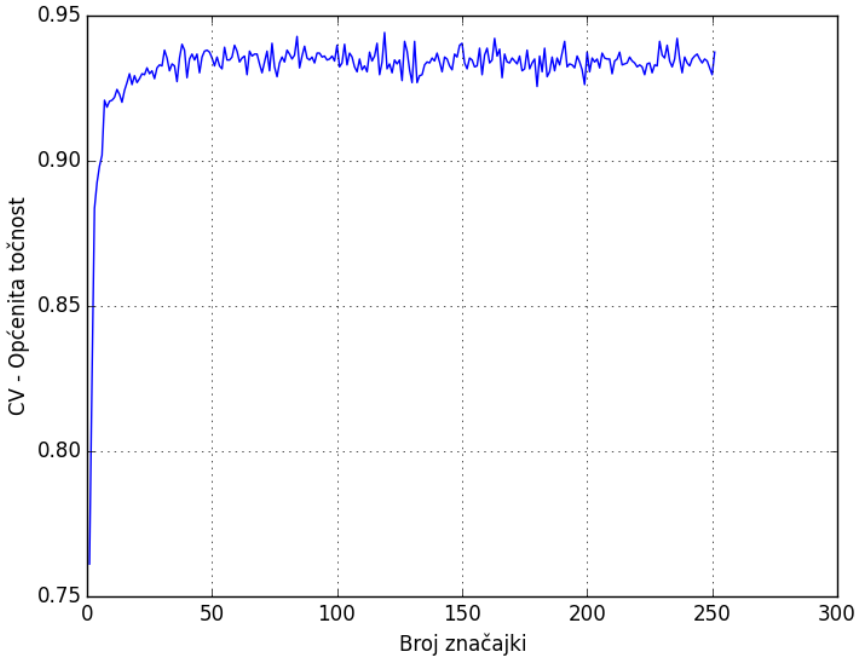
Tablica 5.5
Usporedba rezultata klasifikatora nakon odabira značajki metodom rekurzivne eliminacije(RFE) i redukcije dimenzionalnosti (PCA) za matricu S

Klasifikator	Stopa istinitih pozitiva TPR	Stopa lažnih pozitiva FPR	Stopa pravnih negativa TNR	Stopa lažnih negativa FNR	Općenita točnost
Logistička regresija	0.958600	0.182309	0.817691	0.041400	0.914411
Logistička regresija, RFE (n=51)	0.942336	0.277238	0.722762	0.057664	0.873478
Logistička regresija, PCA (n=58)	0.956629	0.190939	0.809061	0.043371	0.910352
SVM	0.972400	0.110032	0.889968	0.027600	0.946549
SVM, RFE (n=51)	0.966486	0.152104	0.847896	0.033514	0.929296
SVM, PCA (n=58)	0.989157	0.361381	0.638619	0.010843	0.879229
Metoda nasumične šume	0.987679	0.098166	0.901834	0.012321	0.960758
Metoda nasumične šume, RFE (n=51)	0.987186	0.093851	0.906149	0.012814	**0.961773**
Metoda nasumične šume, PCA (n=58)	0.983243	0.141316	0.858684	0.016757	**0.944181**

Prilikom korištenja metode glavnih komponenata PCA uvjet za odabir broja komponenti

bio je da izabrane glavne komponente zadržavaju minimalno 95% varijance originalnog skupa značajki. Prije primjene same metode i računanja nove reducirane matrice originalni podaci su standardizirani kako bi njihovo rasipanje odgovaralo normalnoj razdiobi. Uz uvjet zadržavanja 95% varijance početnog skupa matrica S je reducirana na 58 glavnih komponenti, matrica $D1$ je reducirana na 122 glavne komponente, a kao što smo i napomenuli matrica $D2$ je reducirana na 50 glavnih komponenti.

Primjenom redukcije dimenzionalnosti, metodom PCA na matrici S, primjenjive i usporedive točnosti s cjelovitim skupom značajki postignute su korištenjem metoda logističke regresije ili nasumične šume (tablica 5.5). Metoda potpornih vektora (SVM) imala je značajan rast lažnih pozitiva (s 11% na 36%), dok je metoda nasumične šume imala prihvatljivi rast stope lažnih pozitiva s 9% na 14%. Logistička regresija s reduciranim skupom od 58 glavne komponente imala je približno jednaku općenitu točnost kao i prilikom korištenjem cjelovitog skupa značajki. Metode logističke regresije i nasumične šume imaju prihvatljive klasifikacijske rezultate prilikom korištenja redukcije dimenzionalnosti metodom glavnih komponenta te se mogu koristi u osnovnom klasifikatoru C_S.



Slika 5.7: Utjecaj redukcije broja značajki metodom rekurzivne eliminacije na općenitu točnost (matrica $D1$). Model s 119 značajke postiže općenitu točnost 0.9441

Cjelovita matrica $D1$ se sastoji od 252 značajke. Prilikom eksperimenta odabrane su 119 najbolje rangirane značajke koje postižu najveću općenitu točnost od 94,49% (slika 5.7). U tablici 5.6 dani su rezultati klasifikatora s cjelovitim i reduciranim skupovima značajki. Testirani klasifikatori postigli su usporedive rezultate s reduciranim skupom značajki (n=119) osim u slučaju metode potpornih vektora, čiji je broj lažnih pozitiva porastao na neprihvatljivu razinu (69,57%). U hibridnom klasifikatoru može se koristiti logistička regresija ili metoda nasumične šume kao klasifikator C_{D1} s reduciranim skupom od 119 značajke izabrane metodom rekurzivne eliminacije. Logistička regresija postiže s reduciranim skupom značajki slične rezultate kao i s cjelovitim skupom značajki (općenita točnost od 93%), dok metoda nasumične šume ima neznatno smanjenje općenite točnosti s 95% na 94% uz podjednak broj lažnih pozitiva.

Tablica 5.6
Usporedba rezultata klasifikatora nakon odabira značajki metodom rekurzivne eliminacije (RFE) i redukcije dimenzionalnosti (PCA) za matricu $D1$

Klasifikator	Stopa istinitih pozitiva TPR	Stopa lažnih pozitiva FPR	Stopa pravnih negativa TNR	Stopa lažnih negativa FNR	Općenita točnost
Logistička regresija	0.950222	0.105717	0.894283	0.049778	0.932679
Logistička regresija, RFE (n=119)	0.951207	0.105717	0.894283	0.048793	0.933356
Logistička regresija, PCA (n=122)	0.942829	0.172600	0.827400	0.057171	0.906631
SVM	0.965993	0.078749	0.921251	0.034007	0.951962
SVM, RFE (n=119)	0.988664	0.695793	0.304207	0.011336	0.774019
SVM, PCA (n=122)	0.976343	0.330097	0.669903	0.023657	0.880244
Metoda nasumične šume	0.970429	0.086300	0.913700	0.029571	0.952639
Metoda nasumične šume, RFE (n=119)	0.968950	0.093851	0.906149	0.031050	**0.949256**
Metoda nasumične šume, PCA (n=122)	0.968457	0.220065	0.779935	0.031543	**0.909337**

Svi testirani klasifikatori s PCA reduciranom matricom $D1$ imali su smanjenu općenitu točnost uz povećanje stope lažnih pozitiva (tablica 5.6) u odnosnu na korištenje matrice sa svim značajkama. Logistička regresija imala je najmanje smanjenje općenite točnosti, s 93% na 90%. Metoda potpornih vektora SVM, kao i u slučaju korištenja odabira značajki metodom RFE, nije postigla usporedive rezultate s cjelovitim skupom značajki zbog neprihvatljivo velike stope lažnih pozitiva (za PCA FPR=33%). Metoda nasumične šume imala značajan rast stope lažnih pozitiva, s 9% na 22%. Prilikom korištenja metode glavnih komponenta na matrici $D1$ prema rezultatima kao osnovni klasifikator C_{D1} preporučljivo je koristiti logističku regresiju zbog najmanjeg utjecaja na rezultate u odnosu na rezultate dobivene korištenjem cjelovitog skupa značajki.

Pregled izabranih značajki metodom rekurzivne eliminacije za matrice S i $D1$ dan je u prilogu C.

5.5 Vremensko trajanje analiza

U ovom poglavlju prikazano je vremensko trajanje izvršavanja svake pojedinačne operacije okvira Franko. Vrijeme trajanja je izuzeto iz dnevničkih zapisa koje okvir Franko zapisuje. Iz zapisa su uzete razlike između dva relevantna događaja za svaku promatranu operaciju. Pošto ni jedna operacija nije paralelizirana, odnosno sve se izvode serijski, ovo je predstavljao najbrži način izračunavanja vremenskog trajanja pojedinačnih operacija koje okvir Franko izvodi te njihovih pripadajućih statistika.

Glavne operacije okvira Franko koje su obrađene u ovom poglavlju su sljedeće :

(a) Vremensko trajanje obrade izvještaja modulom *collector.py*. Operacija uključuje prilagođenu statičku analizu te dohvat i obradu sirovnih podataka koji se spremaju u relacijski model, a rezultat su dinamičke analize Cuckoo Sandboxom.

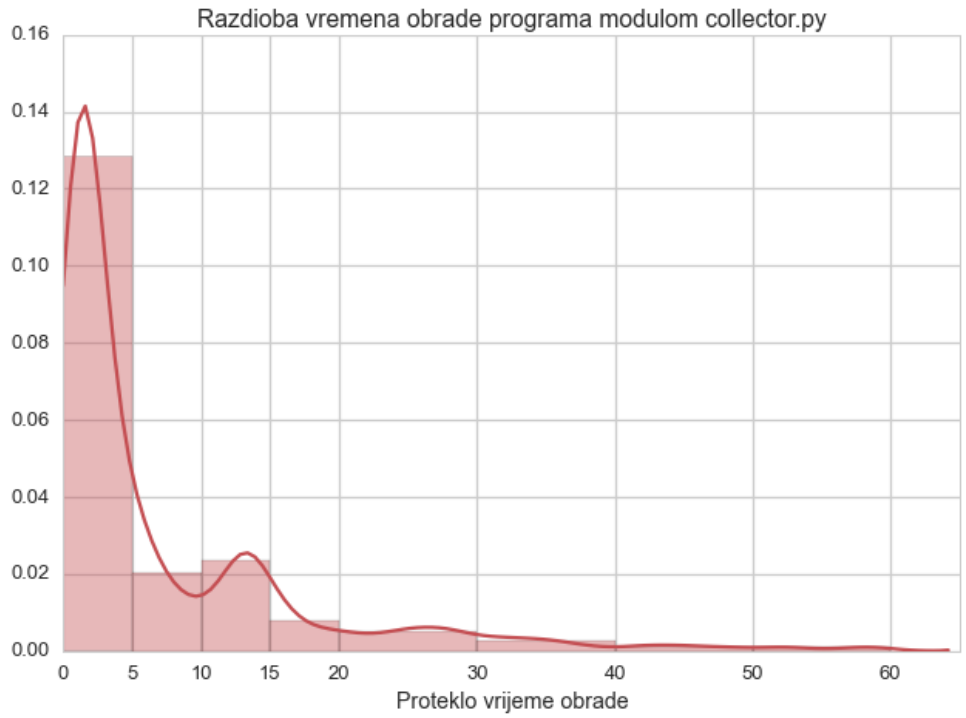
(b) Vremensko trajanje stvaranja statičkih značajki iz relacijske baze podataka okvira Franko.

(c) Vremensko trajanje stvaranja dinamičkih značajki iz relacijske baze podataka okvira Franko.

Tablica 5.7
Statistika vremenskog trajanja modula *collector.py* na skupu TRAIN-1

	Vremensko trajanje (u sek.)
Ukupno programa	2955
Aritmetička sredina	7,753
Standardna devijacija	11,569
Minimalno trajanje	0,713
I kvartil	1,132500
II kvartil (Medijan)	2,395
III kvartil	10,877000
Maksimalno trajanje	122,822

Prosječno vrijeme trajanja izvršavanja modula *collector.py* iznosi 7,75 sekundi. Ovaj modul je vremenski najzahtjevniji jer dohvaća i obrađuje izvještaj analize Cuckoo Sandboxa te ga prilagođava relacijskom modelu i provodi statičku analizu programa. U prosjeku trajanje obrade za većinu uzoraka iznosi manje od 5 sekundi. Maksimalno vrijeme trajanja obrade zabilježeno na skupu TRAIN-1 iznosi 122 sekunde. Ovakve duge obrade programa uzrokuje veliki broj generiranih procesa i poziva prema operacijskom sustavu prilikom dinamičke analize. Neki od analiziranih programa najčešće koriste tehniku zaobilaženja mehanizma detekcije (npr. metodom *anti-vm*), kojom se nakon prepoznavanja virtualne okoline generira mnogo beskorisnih događaja s ciljem zagušenja obrade rezultata analize. Na sličan način autori zlonamjernih programa otežaju statičku analizu dodavanjem beskorisnih podataka u PE strukturu. Prema rezultatima za operaciju obrade izvještaja (*collector.py*), prikazanim u tablici 5.7 i razdiobi vremenskog trajanja prikazanoj na slici 5.8, medijan za skup TRAIN-1 iznosi 2,39 sekundi, dok obrada za 75% programa traje do 10,87 sekundi. Ubrzanje pojedinačne analize, ne uzimajući u obzir paralelizaciju, moguće je uvođenjem koraka koji prilikom spremanja izvještaja nakon dinamičke analize prepoznaju korištene napadačke tehnike te stvaraju sažetak njihovih rezultata (npr. uzimanjem uzorka poziva).



Slika 5.8: Razdioba vremenskog trajanja obrade modulom *collector.py* - uključuje obradu programa statičkom analizom i pretvaranje izvještaja u pogodan format za unos u bazu podataka

Vrijeme potrebno da se iz relacijske baze podataka okvira Franko generiraju statičke značajke u prosjeku traje 0,01 sekunde, dok maksimalno vrijeme trajanje iznosi 0,065 sekundi. Ovo vrijeme trajanja je zanemarivo za samu obradu. Statistike vezane za dohvat i generiranje statičkih značajki moguće je vidjeti u tablici 5.8.

Tablica 5.8
Statistika vremenskog trajanja stvaranja statičkih značajki na skupu TRAIN-1

	Vremensko trajanje (u sek.)
Ukupno programa	2955
Aritmetička sredina	0,018
Standardna devijacija	0,003
Minimalno trajanje	0,011
I kvartil	0,017000
II kvartil (Medijan)	0,019
III kvartil	0,020000
Maksimalno trajanje	0,065

Zbog količine pohranjenih podataka izvlačenje dinamičkih značajki traje nešto duže, u prosjeku 2,44 sekunde, kod 75% programa koji su dio skupa TRAIN-1 zabilježeno je vremensko trajanje do 2,66 sekundi potrebno da se izvuku dinamičke značajke korištene

u matricama $D1$ i $D2$. Rješavanjem problematičnih analiza, kako je opisano kod analize trajanja obrade izvještaja, moguće je dodatno smanjiti vremensko trajanje ove operacije.

Tablica 5.9
Statistika vremenskog trajanja stvaranja dinamičkih značajki na skupu TRAIN-1

	Vremensko trajanje (u sek.)
Ukupno podatka	2955
Aritmetička sredina	2,448
Standardna devijacija	0,755
Minimalno trajanje	0,002
I kvartil	2,597
II kvartil (Medijan)	2,612
III kvartil	2,663
Maksimalno trajanje	3,118

U prosjeku vrijeme provedeno za obradu pojedinačnog progama, uzmemo li aritmetičke sredine obrade, iznosi 10,2 sekunde na prosječnom osobnom računalu korištenom za provođenje eksperimenata. Obrada i pohrana u relacijsku bazu podataka predstavlja najduži proces, osim same obrade u okruženju Cuckoo Sandbox. Relacijska baza podataka se koristi zbog praktičnosti, stabilnosti i autorovog poznavanja iste, ali se sama može izostaviti te se značajke mogu direktno generirati iz baze podataka MongoDB koju Cuckoo Sandbox koristi.

Napomenimo, da u sam proces obrade programa nije uključeno vremensko trajanje dinamičke analize čije maksimalno trajanje iznosi 120 sekundi. Za provođenje dinamičke analize korištenjem okoline Cuckoo Sandbox 1.3 korišten je poslužitelj sljedeće konfiguracije: dva procesora Xeon E5405 na 2Ghz i 20GB radne memorije. Korišteni poslužitelj star je preko 8 godina te bi samo korištenje novijeg hardvera, poput SSD diskova, drastično smanjilo trajanje analiza te se neki postupci mogu optimirati tj. dodatno paralelizirati. Na stroju se nalazio hipervizor VirtualBox s deset virtualnih strojeva, od kojih je svaki koristio operacijski sustav Windows 7 i 1GB radne memorije. U jednom satu se u prosjeku uspješno obradi do 250 programa, što odgovara mogućnošću obrade do 5280 programa unutar jednog dana. Vremenske uštede u ovom segmentu, izuzmemo li povećanje strojeva za obradu, moguće je ostvariti korištenjem drugog hipervizora, nativnog KVM, ali od njegovog korištenja se odustalo u početku istraživanju zbog nestabilnosti i loše podrške koju je tada nudila trenutna verzija Cuckoo Sandboxa. Valja napomenuti da KVM troši manje resursa prilikom pokretanja virtualnog stroja i omogućava u praksi veću paralelizaciju i fleksibilnost u usporedbi s hipervizorom VirtualBox. Također, u početnim fazama istraživanja koristio se operacijski sustav Windows XP čiji su virtualni strojevi koristili manje

resursa, ali zbog stvarnijih uvjeta eksperimenta i novih mogućnosti aktualnih zlonamjernih programa virtualni strojevi su nadograđeni na verziju Windows 7.

5.6 Rezultati hibridnog klasifikatora

Hibridni klasifikator sastavljen je od rezultata klasifikatora prvog reda C_S, C_{D1} i C_{D2}. Svaki od klasifikatora daje dvije vjerojatnosti, prva predstavlja vjerojatnost da je program benigni $P(C/B)$ dok je druga vjerojatnost da je program maliciozan $P(C/M)$. Za ove vjerojatnosti vrijedi $P(C/M) + P(C/B) = 1$. Pošto u hibridnoj metodi koristimo tri klasifikatora 0-reda to čini ukupno šest značajki koje koristimo za dobivanje konačne klasifikacijske oznake. Skup s vjerojatnosnim značajkama pripadnosti pojedinoj klasi označavamo s HP. Također, u eksperimentu su prikazane usporedbe rezultata hibridnog klasifikatora koji koristi značajke s diskretnim klasifikacijama HB. Prilikom korištenja skupa HB ne uzimaju se u obzir vjerojatnosti pripadnosti, već klase koje su izlaz klasifikatora 0-reda, odnosno svaka od tri značajke nam govori je li pojedini klasifikator program klasificirao kao maliciozan ili benigni.

Tablica 5.10
Rezultati hibridnog klasifikatora na skupu za učenje TRAIN-1

Skup	Klasifikator	Stopa ispravnih pozitiva TPR	Stopa lažnih pozitiva FPR	Stopa pravnih negativa TNR	Stopa lažnih negativa FNR	Općenita točnost
HP	Logistička regresija	0.985214	0.017260	0.982740	0.014786	0.984438
HP	Stablo odlučivanja C 4.5	0.987186	0.023732	0.976268	0.012814	0.983762
HP	**Metoda nasumične šume**	**0.988172**	**0.015102**	0.984898	0.011828	**0.987145**
HB	Logistička regresija	0.983243	0.014024	0.985976	0.016757	0.984100
HB	Stablo odlučivanja C 4.5	0.983243	0.014024	0.985976	0.016757	0.984100
HB	**Metoda nasumične šume**	**0.983243**	**0.014024**	0.985976	0.016757	**0.984100**
HB	Metoda glasovanja	0.977822	0.042071	0.957929	0.022178	0.97158

U tablici 5.10 prikazana je usporedba korištenja različitih metoda strojnog učenja kod hibridnog klasifikatora za skup za učenje TRAIN-1. Tri testirane metode: logistička regresija, stablo odlučivanja i metode nasumične šume, postižu slične rezultate na skupu za učenje TRAIN-1. Za skup HP najveću općenitu točnost 98,71% i najmanju stopu lažnih pozitiva ima metoda nasumične šume (1,51%), što je bolje od općenitih točnosti koji postižu klasifikatori 0-reda: C_S (95,97%), C_{D1} (95,06%) i C_{D2} (91,91%). Za konačni klasifikator 1-reda izabrana je metoda nasumične šume prvenstveno zbog toga što ima mogućnost dobrog obuhvaćanja grešaka koje čine osnovni klasifikatori , a istu metodu koriste i svi osnovni klasifikatori. Poznato je da metoda nasumične šume zbog korištenja više različitih stabla odlučivanja i agregacije njihovih rezultat postiže dobre rezultate kod nelinearnih klasifikacijskih problema, što se pokazalo prikladnim za primjenu u hibridnom klasifikatoru.

Korištenjem skupa značajki HB najveća općenita točnost je 98,41%, odnosno neznatno je manja od općenite točnosti dobivene korištenjem skupa HP i metode nasumične šume (98,71%). Tingovo i Wittenovo istraživanje [99] preporučuje upravo korištenje vjerojatnosti prilikom slaganja klasifikatora jer prema njihovim eksperimentima u pravilu se postižu bolji rezultati klasifikacije od primjene diskretnih klasa.

Metoda glasovanja postigla je općenitu točnost od 97,15% uz stopu lažnih pozitiva od 4,2%, što je lošiji rezultat od nove hibridne metode. S obzirom na rezultate pokazalo se opravdano koristiti različite skupove značajki i slaganje klasifikatora kako bi se povećala točnost klasifikacije u odnosnu na pojedinačne (osnovne) klasifkatore.

5.6.1 Rezultati hibridnog klasifikatora na neovisnom skupu

Neovisni ili testni skup TEST-1 sadržava 638 malicioznih programa koji su se pojavili u tijeku 2016. godine. Neovisni skup prikupljen je iz baze malicioznih programa Malekal Malware Database te sadržava maliciozne programe koje poznati antivirusni programi nisu mogli prepoznati kada su se pojavili. Baza malicioznih programa Malekal korištena je ne samo zbog ažurnosti uzoraka malicioznih programa, već i zbog prisutnih rezultata na servisu VirusTotal u trenutku pojave malicioznog programa, koji znaju ukazivati na nemogućnost detekcije klasičnim antivirusnim alatima. Od pedesetak antivirusnih alata koje koristi servis Virustotal, u pravilu njih 5-30% prepoznaje novi maliciozni program kada se pojavi u bazi Malekal. Za usporedbu, antivirusni alat Avast je od izuzetog neovisnog skupa malicioznih programa TEST-1 ispravno prepoznao samo 36% programa, dok je antivirusni alat Kaspersky, koji prednjači po klasifikacijskim mogućnostima u odnosu na druge alate, prepoznao je ukupno 86% malicioznih programa iz skupa TEST-1.

Neovisnom skupu pridodano je 67 benignih programa izuzetih iz operacijskog sustava

Windows 10 koji nisu bili korišteni u skupu za učenje. Rezultati klasifikacije na uzorcima iz neovisnog skupa nalaze se u tablici 5.11. U tablicu 5.11 su zbog mogućnosti usporedbe uvrštene točnosti klasifikacije antivirusnim alatima Avast i Kaspersky, uz pretpostavku da prepoznaju ispravno sve korištene benigne programe. Hibridni klasifikator ispravno je prepoznao ukupno 98,5% malicioznih programa uz niti jedan lažni pozitiv, a samo 8 malicioznih programa je krivo klasificirano. Klasifikator C_S ima slične rezultate kao i C_H, a od C_H ga razlikuje krivo prepoznat maliciozni program (FPR=1,49%). Ukoliko hibridni klasifikator usporedimo s metodom glasovanja, koja je prepoznala 93% malicioznih programa te pri tome je krivo označila 39 benignih programa (FNR=6,9%), vidljivo je da hibridna metoda postiže bolje rezultate.

Tablica 5.11
Rezultati hibridnog klasifikatora na neovisnom skupu za testiranje TEST-1

Skup	Klasifikator	Stopa ispravnih pozitiva TPR	Stopa lažnih pozitiva FPR	Stopa pravnih negativa TNR	Stopa lažnih negativa FNR	Općenita točnost
PE	Avast AV	0,36	0	1	0,64	0,42
PE	Kaspersky AV	0,86	0	1	0,14	0,87
S	Metoda nasumične šume	0,987589	0,014925	0,985075	0,012411	0,987322
D1	Metoda nasumične šume	0,925532	0	1	0,074468	0,933439
D2	Metoda nasumične šume	0,904255	0,014925	0,985075	0.095745	0,912837
HP	Hibridni klasifikator	0,985816	0	1	0,014184	0,987322
HP	Hibridni klasifikator (korištena RFE za C_S n=51 i C_{D1} n=119)	0,960993	0	1	0,039007	0,965135
HP	Hibridni klasifikator (korišten PCA za C_S n=58 i C_{D1} n=122)	0,930851	0	1	0,069149	0,938193
HB	Metoda glasovanja	0,930851	0	1	0,069149	0,938193

Ograničenje prve hipoteze je da nova metoda ne otpakirava ili disasemblirala izvršne programe, kao što je opisano u samoj metodologiji istraživanja i postupku izrade hibridne

metode. Korišteni osnovni klasifikatori poštovali su dana ograničenja u H1. Na temelju dobivenih rezultata na skupu TEST-1 (opć. točnost 98,58%, FPR=0%) možemo prihvatiti prvu hipotezu pošto općenita točnost prelazi 90%, a stopa lažnih pozitiva je ispod 10%. Također, možemo prihvatiti i drugu hipotezu jer hibridni klasifikator postiže veću općenitu točnost od svakog pojedinog klasifikatora prvog reda, a ukoliko općenita točnost je jednaka nekom od osnovnih klasifikatora hibridni klasifikator smanjuje stopu lažnih pozitiva. Prilikom korištenja skupa TRAIN-1 hibridni klasifikator je postigao bolje rezultate od svakog pojedinačnog klasifikatora C_S, C_{D1} i C_{D2} (poglavlje 5.6) dok je u slučaju korištenja skupa TEST-1 općenita točnost jednaka klasifikatoru C_S (98,73%) uz redukciju broja lažnih pozitiva.

5.6.2 Usporedba hibridnog algoritma sa sličnim istraživanjima

Usporedimo li hibridni klasifikator sa sličnim istraživanjima (tablica 5.12), koja ujedno predstavljaju trenutno dostignuće znanosti u domeni prepoznavanja zlonamjernih programa, možemo zaključiti da je metoda konkurentna i postiže slične ili značajno bolje rezultate nego slična relevantna istraživanja. Iako Andersonovo istraživanje [7] koristi značajno veći neovisni skup valja napomenuti da su neovisnom skupu TEST-1 pridodani benigni programi koji su bili obuhvaćeni u skupu za učenje, što nije bio slučaj u njegovom istraživanju. Također, stopa pravih pozitiva od 98,58% nadmašuje rezultate koje postižu antivirusni alati Avast (36%) i Kaspersky (86%), a koji predstavljaju standard za detekciju nepoznatih malicioznih programa zbog svojih kvalitetnih heuristika. Rezultati antivirusnih alata na skupu TEST-1 vidljivi su u tablici 5.11.

Istraživanje Santosa i suradnika [82] ne daje procjenu uspješnosti klasifikatora na neovisnom skupu, ali ima visoku općenitu točnost od 96,22% na skupu za učenje. Sličan je slučaj kod istraživanja Islama i suradnika koje proces učenja i testiranja točnosti provode na istom skupu te njihova metoda ima općenitu točnost od 97,055%. Rieckova metoda [76] ima ograničenje što prilikom detekcije maliciozne programe pridodaje postojećim familijama zlonamjernih programa, njegova metoda ima općenitu točnost od 76% na neovisnom skupu od 520 malicioznih i 498 benignih programa te 88% na skupu za učenje.

Tablica 5.12
Rezultati sličnih istraživanja koji koriste hibridne značajke

Istraživanje	Općenita točnost	Broj uzoraka
Anderson i suradnici [7]	97,7%	20,036 malicioznih programa
Santos i suradnici [82]	96,22%	CV skup, 1000 malicioznih i 1000 benignih programa
Islam i suradinici [40]	97,055%	CV skup, 2398 malicioznih i 541 benigna programa
Rieck i suradnici [76] (Koristi se samo dinamički model)	76%	520 malicioznih i 498 benignih programa
Hibridni klasifikator	**98,73%**	638 malicioznih i 67 benignih programa

5.7 Osvrt na hipoteze

Pretpostavka novog hibridnog klasifikatora je da se ne oslanja na potpise antivirusnih alata, već klasifikaciju temelji na metodama nadziranog učenja. Hibridni klasifikator je uspoređen s relevantnim metodama koje su se pojavile prethodnih godina i koje su koristile hibridne, statičke i dinamičke, značajke za klasifikaciju prirode programa. Usporedbom rezultata relevantnih istraživanja [6] [40] [82] [76] u prvoj hipotezi dana je sljedeća pretpostavka o točnosti rezultata klasifikacije koju bi nova metoda trebala zadovoljiti, a koja je svojstvena za sve slične metode:

H1. Nova metoda ima općenitu točnost klasifikacije, bez prethodnog otpakiravanja ili disasembliranja programskog koda, veću od 90% te stopu lažnih pozitiva manju od 10%.

U poglavlju 5.6 se nalaze rezultati hibridne metode na neovisnom skupu TEST-1 koji sadrži zlonamjerne programe vremenski i sadržajno neovisne od skupa za učenje. Većina metoda, osim one opisane u Andersonovom istraživanju [6] [7] i Rieckovom [76] istraživanju, nije koristilo neovisni skup za usporedbu rezultata. Na temelju rezultata dobivenih korištenjem skupa za učenje TRAIN-1, gdje se koristila unakrsna validacija, postignuti su sljedeći rezultati:

- TPR=98,9%
- FPR=1,5%
- TNR=98,5%
- FNR=1,1%
- Općenita točnost 98,7%.

Na neovisnom skupu TEST-1 postignuti su rezultati:

- TPR=98,5%
- FPR=0%
- TNR=1%
- FNR=1,4%
- Općenita točnost 98,7%.

Redukcijom značajki metodom rekurzivne eliminacije postignuti su isto tako vrlo dobri rezultati (općenita točnost 96,5%). Hibridni klasifikator prikazan u istraživanju, za razliku od Andersonove metode [7], ne koristi metodu disasembliranja programa zbog poštivanja pravnih ograničenja prisutnih kod benignih programa, opisani u poglavlju 1.5, te se programi ne otpakiravaju zbog složenosti automatizacije samog postupka bez intervencije analitičara, kao što je to slučaj u istraživanju Islam [40].

Iz prikazanih rezultata moguće je zaključiti da razvijeni hibridni klasifikator na temelju prikupljenih skupova za učenje i testiranje postiže u pravilu bolje rezultate od referentnih istraživanja, prema tome možemo prihvatiti hipotezu H1.

Druga hipoteza istraživanja temelji se na pretpostavci da će odluka donesena s više klasifikatora različitih vrsta u konačnici biti točnija nego korištenjem zasebnih (pojedinačnih) klasifikatora.

H2. Točnost klasifikacije zlonamjernih programa korištenjem nove metode bit će veća od točnosti pojedinačnog klasifikatora koji koristi značajke samo statičke ili dinamičke analize.

Usporedimo li općenite točnosti dobivene na skupu TRAIN-1 za osnovne klasifikatore ($C_S = 95,9\%$, $C_{D1} = 95\%$ i $C_{D2} = 93\%$) s točnošću hibridne metode $C_H = 98,7\%$, možemo zaključiti da hibridna metoda postiže bolje rezultate nego pojedinačni (osnovni) klasifikatori. Na neovisnom skupu rezulatati su bili sljedeći: $C_S = 98,7\%$, $C_{D1} = 93,3\%$, $C_{D2} = 91,2\%$ i $C_H = 98,7\%$. Rezultati na skupu TEST-1 pokazuju da je općenita točnost jednaka za klasifikatore C_S i C_H, s razlikom da hibridni klasifikator ispravno detektira lažni pozitiv koji je pristupan kod klasifikatora C_S. Metoda slaganja osnovnih klasifikatora, korištena kod hibridnog klasifikatora, omogućuje proširenje modela detekcije novim klasifikatorima te je na neovisnom skupu TEST-1 pokazala bolje rezultate nego aktualni antivirusni alati koji postižu općenitu točnost do 87% (KasperskyAV). Na temelju rezultata možemo prihvatiti hipotezu H2.

Poglavlje 6

Zaključak

Istraživanje prikazano u ovoj disertaciji imalo je za cilj razviti novu metodu otkrivanja zlonamjernih programa koja za potrebe klasifikacije programa u zlonamjerne ili dobronamjerne koristi značajke dobivene statičkom i dinamičkom analizom. Razvijena hibridna metoda ima mogućnost detekcije zlonamjernih programa, koristeći za to algoritme nadziranog učenja, te ima mogućnost detekcije prethodno nepoznatih programa.

Hibridna metoda predstavlja novi pristup rješavanju problema prepoznavanja zlonamjernih programa jer kombinira pojedinačne klasifikatore koji koriste značajke iz različitih tipova analiza u svrhu bolje klasifikacije programa. Hibridna metoda ima mogućnost proširivanja novim osnovnim klasifikatorima koji se koriste za donošenje konačne odluke. Slična istraživanja Santosa [82] i Islama [40] ne kombiniraju klasifikatore za donošenje odluke, već kombiniraju značajke različitih vrsta analiza. Također, metodu slaganja klasifikatora nitko do sada nije koristio u domeni detekcije zlonamjernih programa te nisu bili poznati rezultati i mogućnosti primjene metode slaganja klasifikatora do ovog istraživanja. Nadalje, doprinos istraživanja je korištenje značajki koje do sada nisu bile korištene u sličnim, referentnim istraživanja [6] [82] [40]. Primjerice, kod statičkih značajki S provjerava se integriret sadržanih adresa u izvršnoj datoteci, tj. postoje li nelogičnosti prilikom poravnanja u memoriji te se provjerava korištenje sumnjivih funkcija koje maliciozni programi inače koriste za tehnike skrivanja prilikom njihove analize. Dinamičke značajke D_1 sadržavaju podatke vezane za kategoriju poziva prema operacijskom sustavu poput: omjera uspješnosti poziva, njihovo vremensko trajanje, statističke podatke korištenih argumenata u pozivu i sl. Doprinos ovog istraživanja je i smanjenje broja primjera za učenje korištenjem stratificiranog uzorka, koji bolje odražava različitost malicioznih programa, i traženje optimalnih parmetara metoda strojnog učenja. Stratificirani uzorak izabran je prema familijama zlonamjernih programa te sadržava programe iz razdoblja od 2011.-2016., dok je neovisan skup sadržavao aktualne zlonamjerne programe iz 2016. godine. U sličnim istraživanja nije se posvećivala posebna pažnja izradi i izboru skupa za učenje te su izostala testiranja klasifikacije točnosti na neovisnom skupu.

Ograničenje ovog istraživanja je što u domeni otkrivanja zlonamjernih programa ne postoji univerzalni skup za testiranje i usporedbu rezultata novih metoda prepoznavanja malicioznih programa. Razvijena hibridna metoda u eksperimentima na neovisnom skupu postiže bolje rezultate od komercijalno dostupnih antivirusnih alata, što ne znači da je zamjena za njih, već da se ona može upotrijebiti za prepoznavanje novih i prethodno nepoznatih malicioznih programa u mrežnim okolinama gdje postoji mogućnost njihove pojave. Hibridna metoda predstavlja nadopunu postojećim pristupima detekcije te se njena namjena očituje u obradi velikih količina programa i prepoznavanju novih zlonamjernih programa. Za potrebe detekcije ciljanih napada mogu se dodati klasifikatori sa specifičnostima za takve postojeće napade, primjerice moguće je napraviti dodatni klasifikator koji prepoznaje naredbe na sustavima posebne namjene (npr. industrijska postrojenja ili bankarski sustavi).

Stručni doprinos istraživanja sadržan je u implementaciji nove metode u obliku programskog koda koju je moguće ugrađivati u postojeće hodograme analiza programa i sustave za detekciju zlonamjernih programa. Društveni doprinos ovog istraživanja očituje se u rješavanju aktualnog problema detekcije zlonamjernih programa, zbog kojih su mnoge organizacije diljem svijeta pretrpjele štete i velike financijske gubitke. Razvijena metoda otkrivanja zlonamjernih programa, kao što je prethodno navedeno, neovisna je o statičkim potpisima za detekciju zlonamjernih programa, dok u isto vrijeme omogućuje detekciju prethodno nepoznatih zlonamjernih programa, što je prikazano u rezultatima na skupu TEST-1. Implementacijom hibridne metode u mrežno okružje omogućuje se analiza velike količine programa i povećanje razine sigurnosti u organizacijama. U sklopu istraživanja izrađena su dva skupa programa: prvi stratificirani skup zlonamjernih programa i dobronamjernih programa namijenjen učenju parametara pojedinačnih klasifikatora te drugi, testni skup, koji se koristi za usporedbu razvijene metode s drugim referentnim pristupima. Skupovi programa javno su objavljeni u svrhu omogućavanja ponavljanja budućih istraživanja.

U domeni prepoznavanja zlonamjernih programa postoji nekoliko otvorenih problema koji mogu biti predmetom budućih istraživanja, a kojima se ovo istraživanje bavilo u manjoj mjeri ili su nadilazili njegovo opseg, poput:

- Unaprjeđenja otpornosti postojećih zaštićenih okolina na metode sakrivanja i otežavanja analize malicioznih programa.
- Uključivanje iterativnog, vremensko ovisnog, učenja novih karakteristika zlonamjernih programa, kako bi se izbjegao problem slabljenja naučenog koncepta.
- Uključivanje ljudskog analitičara u proces prepoznavanja slabljenja naučenog koncepta i evaluacije modela.

- Dodavanje novih klasifikatora, prvenstveno klasifikatora koji koristi postupak disasembliranja programa.

Jedan od najvećih problema pri dinamičkoj analizi je ograničenost zaštićene okoline, odnosno mogućnost da zlonamjerni program prepozna samu okolinu i odgodi svoje izvršavanje. Zlonamjerni programi traže nove načine kako prepoznati postupke dinamičke analize (metode *anti-vm* i *anti-debug*), stoga je potrebno razviti nove mehanizme otporne na metode detekcije virtualnih okolina. Izmjenom virtualnih okolina i osnaživanjem njihovih postavki moguće je spriječiti metode koje koriste zlonamjerni programi. Također, metode za snimanje poziva prema operacijskom sustavu su postale opće poznate i zlonamjerni programi ih vrlo lako prepoznaju te je potrebno smisliti nove načine kako učiniti snimanje aktivnosti programa neprimjetno i otežati njihovu detekciju.

Pokazalo se da sustavi temeljeni na strojnom učenju ne mogu postojati zasebno. Uključivanjem eksperta u petlju učenja (engl. *human in the loop*) [62] [85] moguće je postići bolje rezultate ranom detekcijom krivih klasifikacija i dodatnim podešavanjem parametara metoda strojnog učenja. Pokazalo se da s vremenom zlonamjerni programi mijenjaju svoja svojstva, tako je u ovom istraživanju primijećena razlika između novijih verzija ransomwarea i zlonamjernih programa koji su se pojavljivali prije nekoliko godina, a sadržani su bili u skupu za učenje. Jedno od mogućih rješenja je iterativno vremensko nadopunjavanje skupa aktualnim primjerima za učenje te ponovno učenje klasifikatora kako bi se prilagodio promjenama u značajkama zlonamjernih programa. Hibridna metoda može se poboljšati dodavanjem novih klasifikatora. Zasigurno u budućem radu će biti dodan novi osnovni klasifikator koji koristi značajke disasembliranih ekvivalenta programa.

Prilog A

Model podataka okvira Franko

Model podataka okvira Franko predstavlja relacijsku bazu podatka koja je sastavljena od 10 tablica. Veze između tablica prikazane su na slici A.1.

Kratak opis tablica i podataka koji se pohranjuju u njih:

Tablica samples Sadrži sve relevatne podatke koje su zapisane u formatu Portable Executable. Tablica se popunjava nakon izvršene statičke analize. Ovi podaci su dobiveni prvenstveno iz PE strukture IMAGE OPTIONAL HEADER [1].

Tablica sample_process Sadrži sve procese koje je analizirani program pokrenuo prilikom dinamičke analize u okruženju Cuckoo Sandbox.

Tablica syscalls Sadrži zabilježene pozive prema operacijskom sustavu analiziranog programa, sama tablica se vezuje tablicu procesa iz razloga što analizirani program može pokrenuti više procesa.

Tablica pe_section Ova tablica sadrži podatke o sekcijama koje program sadrži, podaci se zapisuju nakon statičke analize. Podaci su dobiveni obradom liste iz PE strukture IMAGE SECTION HEADER[2] koja sadrži podatke o sekcijama.

Tablica pe_imports_exports Sadrži podatke o bibliotekama koje program uvozi ili daje drugim programima na korištenje, podaci se zapisuju nakon statičke analize. Podaci se dobivaju obradom liste IMAGE IMPORT DESCRIPTOR iz PE strukture, a koja sadrži veze na uvezene biblioteke te IMAGE EXPORT DIRECTORY za izvezene datoteke. U tablici IMAGE DATA DIRECTORY nalaze se pomaci (ofset)

[1] IMAGE_OPTIONAL_HEADER structure - https://msdn.microsoft.com/en-us/library/windows/desktop/ms680339(v=vs.85).aspx

[2] IMAGE_SECTION_HEADER structure - https://msdn.microsoft.com/en-us/library/windows/desktop/ms680341(v=vs.85).aspx

potrebani da bi se došlo do točnih adresa tablica u memoriji.

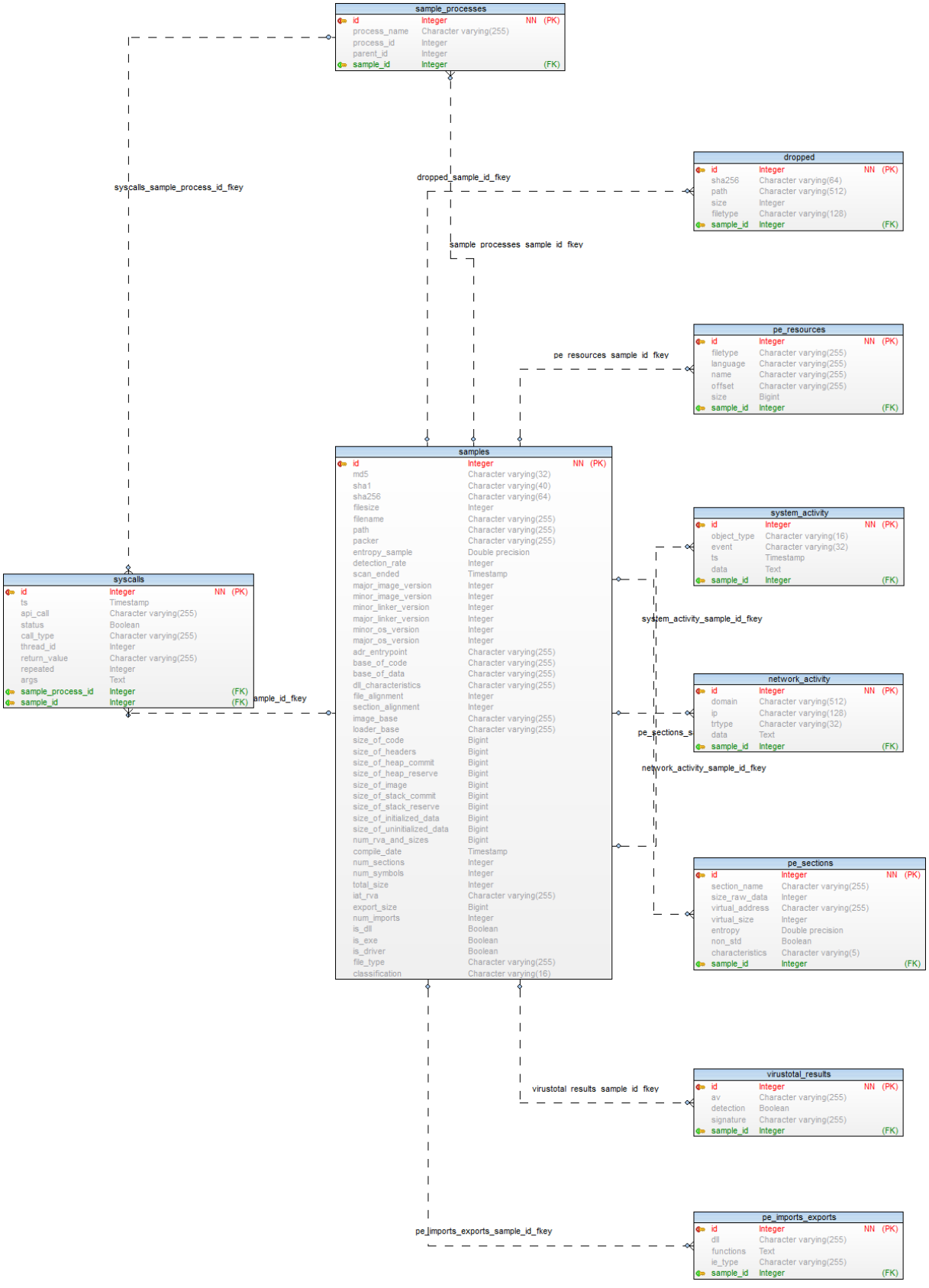
Tablica pe_resources Sadrži podatke o resursima (npr. ikone, slike i sl.) koji su pohranjeni unutar analiziranog programa, podaci se zapisuju nakon izvršene statičke analize. Podaci o resursima se dobivaju obradom vezane liste IMAGE RESOURCES DIRECTORY koja je pohranjena u PE strukturi.

Tablica dropped Sadrži podatke o datotekama koje je analizirani program zapisao na disk prilikom dinamičke analize u okolini Cuckoo Sandbox.

Tablica system_activity Sadrži podatke o aktivnostima prilikom dinamičke analize, ovo je dio izvještaja analize Cuckoo Sandboxa koja predstavlja sažetak aktivnosti odnosno grupirane vrste poziva prema operacijskom sustavu.

Tablice network_activity Sadrži mrežne konekcije inicirane od analiziranog programa u okolini Cuckoo Sandbox.

Tablica virustotal_results Sadrži klasifikacije antivirusnih alata sa servisa VirusTotal.



Slika A.1: Model podataka sustava Franko

Prilog B

Popis poziva koji koriste zlonamjerni programi

Pozivi koji omogućuju prepoznavanje korištenja *debuggera* ili onemogućuju dinamičku analizu programa (engl. *anti-debug*)

CheckRemoteDebugger
DebugActiveProcess
FindWindow
FindWindowA
FindWindowW
GetLastError
GetWindowThreadProcessId
IsDebugged
IsDebuggerPresent
NtCreateThreadEx
NtGlobalFlags
NtSetInformationThread
ZwSetInformationThread
OutputDebugString
NtContinue
GenerateConsoleCtrlEvent
pbIsPresent
Process32First
Process32Next
TerminateProcess ThreadHideFromDebugger
UnhandledExceptionFilter
ZwQueryInformation
NtQuerySystemInformation
SystemKernelDebuggerInformation
RtlQueryProcessDebugInformation

RtlQueryProcessHeapInformation
NtQueryInformationProcess
CreateToolhelp32Snapshot
GetShellWindow
SuspendThread
NtSuspendThread
DbgSetDebugFilterState
NtSetDebugFilterState
CreateProcess
CreateProcessA
CreateProcessW
CreateMutex
GetThreadContext
SetThreadContext

Pozivi koji se koriste prilikom umetanja u druge procese (engl. *process injection*)
NtResumeThread
NtSetInformationThread
RtlCreateUserThread
SetWindowsHookExA
RtDecompressBuffer
NtTerminateThread
NtWaitForSingleObject
RtlCreateUserThread
CreateRemoteThread
WriteProcessMemory

Pozivi koji se koriste prilikom postupka otpakiravanja
NtWriteVirtualMemory
ZwMapViewOfSection
VirtualAlloc
VirtualFree
VirtualProtect
VirtualAllocEx
LoadLibraryA
GetProcAddress

Pozivi koji se koriste za provjeru virtualnih okolina (engl. *anti-VM*)
GetTickCount
GetLocalTime
GetSystemTime
timeGetTime
NtQueryPerformanceCounter
ZwQueryInformation

Pozivi kriptografskih funkcija, koji su svojstveni za zlonamjerne programe *ransomware*
CryptCreateHash
CryptDestroyHash
CryptGetHashParam
CryptAcquireContext
CryptGenKey
CryptExportKey
CryptImportKey
EncryptFile
EncryptFileW

Prilog C

Izabrane značajke metodom rekurzivne eliminacije

Izabrane značajke metodom RFE za matricu S

1. advapi32.dll
2. antidebug
3. antivm
4. bss_virtual_size
5. comctl32.dll
6. data_characteristics
7. data_entropy
8. data_in_sec
9. data_rawsize_virtsize_ratio
10. data_size_raw_data
11. data_virtual_size
12. file_alignment
13. filesize
14. iat_in_sec
15. idata_rawsize_virtsize_ratio
16. idata_virtual_size
17. image_base_has_regular_adr
18. is_exe
19. kernel32.dll
20. major_image_version
21. major_linker_version
22. major_os_version
23. minor_os_version
24. ntdll.dll
25. num_imports
26. num_rva_and_sizes
27. num_symbols
28. oleaut32.dll
29. rdata_characteristics
30. rdata_entropy
31. rdata_rawsize_virtsize_ratio
32. rdata_size_raw_data
33. rsrc_characteristics
34. rsrc_entropy
35. rsrc_rawsize_virtsize_ratio

36. rsrc_size_raw_data
37. shell32.dll
38. size_of_heap_reserve
39. size_of_image
40. size_of_initialized_data
41. size_of_stack_commit
42. size_of_stack_reserve
43. text_characteristics
44. text_entropy
45. text_rawsize_virtsize_ratio
46. text_size_raw_data
47. tls_virtual_size
48. total_size
49. unpack
50. unusual_dll_imports
51. user32.dll

Izabrane značajke metodom RFE za matricu $D1$

1. FindFirstFileExW_ratio
2. GetSystemTimeAsFileTime_ratio
3. GetVolumeInformationByHandleW_ratio
4. LdrGetDllHandle_ratio
5. LdrGetProcedureAddress_ratio
6. LdrLoadDll_ratio
7. NtAllocateVirtualMemory_ratio
8. NtClose_ratio
9. NtCreateFile_ratio
10. NtCreateSection_ratio
11. NtDeviceIoControlFile_ratio
12. NtDuplicateObject_ratio
13. NtEnumerateValueKey_ratio
14. NtOpenFile_ratio
15. NtOpenKey_ratio
16. NtQueryAttributesFile_ratio
17. NtQueryInformationFile_ratio
18. NtQueryValueKey_ratio
19. NtSetInformationFile_ratio
20. NtTerminateProcess_ratio
21. NtUnmapViewOfSection_ratio
22. NtWaitForSingleObject_ratio
23. NtWriteFile_ratio
24. Process32NextW_ratio
25. RegCloseKey_ratio
26. RegEnumKeyExW_ratio
27. RegEnumValueW_ratio
28. RegOpenKeyExA_ratio
29. RegOpenKeyExW_ratio
30. RegQueryValueExA_ratio
31. RegQueryValueExW_ratio
32. SetUnhandledExceptionFilter_ratio
33. VirtualProtectEx_ratio
34. WriteConsoleA_ratio
35. WriteConsoleW_ratio
36. args_max
37. args_mean
38. args_std
39. closesocket_ratio
40. com_args_max
41. com_args_mean
42. com_args_std
43. com_ratio
44. connect_ratio
45. crypto_args_max
46. crypto_args_mean
47. crypto_args_min
48. crypto_ratio
49. crypto_return_success
50. device_args_max

51. device_args_mean
52. device_args_min
53. device_args_std
54. device_ratio
55. device_return_success
56. filesystem_args_max
57. filesystem_args_mean
58. filesystem_args_min
59. filesystem_args_std
60. filesystem_ratio
61. gethostbyname_ratio
62. hooking_args_max
63. hooking_args_mean
64. hooking_args_std
65. hooking_ratio
66. ioctlsocket_ratio
67. misc_args_max
68. misc_args_mean
69. misc_args_min
70. misc_args_std
71. misc_ratio
72. misc_return_success
73. network_args_max
74. network_args_mean
75. network_args_min
76. network_args_std
77. network_ratio
78. network_return_success
79. process_args_max
80. process_args_mean
81. process_args_min
82. process_args_std
83. process_ratio
84. recv_ratio
85. registry_args_max
86. registry_args_mean
87. registry_args_min
88. registry_args_std
89. registry_ratio
90. send_ratio
91. services_args_max
92. services_args_min
93. services_ratio
94. socket_ratio
95. synchronization_args_max
96. synchronization_args_mean
97. synchronization_args_min
98. synchronization_args_std
99. synchronization_ratio
100. synchronization_return_success
101. syscalls_in_1sec
102. syscalls_timediff_mean

103. syscalls_timediff_std
104. system_args_max
105. system_args_min
106. system_args_std
107. system_ratio
108. system_return_success
109. threading_args_max
110. threading_args_mean
111. threading_args_min
112. threading_args_std
113. threading_ratio
114. threading_return_success
115. windows_args_max
116. windows_args_mean
117. windows_args_min
118. windows_args_std
119. windows_ratio

Bibliografija

[1] T. Abou-Assaleh i dr. "N-gram-based detection of new malicious code". *Computer Software and Applications Conference, 2004. COMPSAC 2004. Proceedings of the 28th Annual International*. Sv. 2. IEEE. 2004.

[2] F. Ahmed i dr. "Using Spatio-temporal Information in API Calls with Machine Learning Algorithms for Malware Detection". *Proceedings of the 2Nd ACM Workshop on Security and Artificial Intelligence*. AISec '09. New York, USA: ACM, 2009.

[3] M. Alazab, S. Venkataraman i P. Watters. "Towards Understanding Malware Behaviour by the Extraction of API Calls". *Cybercrime and Trustworthy Computing Workshop (CTC), 2010 Second*. 2010.

[4] A. O. AlienVault. *ParanoidFish - Pafish*. https://www.alienvault.com/blogs/labs-research/hardening-cuckoo-sandbox-against-vm-aware-malware. pristupano 21.6.2016.

[5] G. Amato. *PEframe 5.0.1*. https://github.com/guelfoweb/peframe. pristupano 21.6.2016.

[6] B. Anderson. "Integrating Multiple Data Views for Improved Malware Analysis". (2014).

[7] B. Anderson, C. Storlie i T. Lane. "Improving Malware Classification: Bridging the Static/Dynamic Gap". *Proceedings of the 5th ACM Workshop on Security and Artificial Intelligence*. AISec '12. New York, USA: ACM, 2012.

[8] T. Atlantic. *The Rise of Asymmetric Cyberwarfare*. 2016.

[9] badtrace.com. *rdtsc x86 instruction to detect virtual machines*. http://blog.badtrace.com/post/rdtsc-x86-instruction-to-detect-vms/. pristupano 21.6.2016.

[10] M. Bailey i dr. "Automated Classification and Analysis of Internet Malware". *Proceedings of the 10th International Conference on Recent Advances in Intrusion Detection*. RAID'07. Berlin, Heidelberg: Springer-Verlag, 2007.

[11] U. Bayer. "TTAnalyze: A Tool for Analyzing Malware". Mag. rad. Technical University of Vienna, 2006.

[12] Z. Bazrafshan i dr. "A survey on heuristic malware detection techniques". *5th Conference on Information and Knowledge Technology (IKT)*. IEEE. 2013.

[13] N. Bogunović. *Otkrivanje znanja u skupovima podatka - Materijali s predavanja*. http://www.zemris.fer.hr/predmeti/kdisc/ref1.html. prsitupano 10.4.2017.

[14] N. Bogunović i B. Dalbelo Bašić. *Otkrivanje znanja u skupovima podataka - Multivarijanta analiza*. http://www.zemris.fer.hr/predmeti/kdisc/bojana/biljeske-OZSP-pogl-1-2-3.pdf. pristupano 19.2.2017.

[15] G. Bonfante, M. Kaczmarek, J.-Y. Marion i dr. "Control flow graphs as malware signatures". *International Workshop on the Theory of Computer Viruses*. 2007.

[16] R. R. Branco i Barbosa. "Prevalent Characteristics in Modern Malwares". BlackHat USA 2014. 2014. URL: https://www.blackhat.com/docs/us-14/materials/us-14-Branco-Prevalent-Characteristics-In-Modern-Malware.pdf.

[17] R. R. Branco, G. N. Barbosa i P. D. Neto. "Scientific but not academical overview of malware anti-debugging, anti-disassembly and anti-vm technologies". BlackHat 2012. 2012.

[18] L. Breiman. "Random forests". *UC Berkeley TR567* (1999).

[19] D. Brumley i dr. "Bitscope: Automatically dissecting malicious binaries". *School of Computer Science, Carnegie Mellon University, Tech. Rep. CMU-CS-07-133* (2007).

[20] E. Carrera i G. Erdélyi. "Digital genome mapping–advanced binary malware analysis". *Virus Bulletin Conference*. 2004.

[21] S. Cesare i Y. Xiang. "Classification of malware using structured control flow". *Proceedings of the Eighth Australasian Symposium on Parallel and Distributed Computing-Volume 107*. Australian Computer Society, Inc. 2010.

[22] S. Cesare i Y. Xiang. "Software Similarity Searching and Classification". *Software Similarity and Classification*. Springer, 2012.

[23] D. Chau i dr. "Polonium: Tera-scale graph mining and inference for malware detection". *SIAM International Conference on Data Mining*. Sv. 2. 2011.

[24] Cuckoo Foundation. *Cuckoo Sandbox: Automated Malware Analysis*. http://www.cuckoosandbox.org/. pristupano 10.9.2014.

[25] A. L. David Harely. *Heuristic analysis: Detecting Unknown Viruses*. http://www.eset.com/us/resources/white-papers/Heuristic_Analysis.pdf. pristupano 15.4.2015.

[26] S. L. documentation. *Tuning the hyper-parameters of an estimator*. http://scikit-learn.org/stable/modules/grid_search.html. pristupano 27.4.2017.

[27] M. Egele i dr. "A survey on automated dynamic malware-analysis techniques and tools". *ACM Computing Surveys* 44.2 (2012).

[28] O. Ferrand. "How to detect the Cuckoo Sandbox and to Strengthen it?": *Journal of Computer Virology and Hacking Techniques* 11.1 (2015), str. 51–58. URL: http://dx.doi.org/10.1007/s11416-014-0224-9.

[29] P. Flach. *Machine Learning: The Art and Science of Algorithms That Make Sense of Data*. New York, USA: Cambridge University Press, 2012.

[30] F. Galton. "Vox populi". *Nature* 75.7 (1907), str. 450–451.

[31] P. M. Granitto i dr. "Recursive feature elimination with random forest for PTR-MS analysis of agroindustrial products". *Chemometrics and Intelligent Laboratory Systems* 83.2 (2006), str. 83–90.

[32] T. Grzinic. *Github Repository - Dataset used for training the hybrid method*. https://github.com/tgrzinic/phd-dataset. pristupano 3.1.2018.

[33] I. Guyon i dr. "Gene selection for cancer classification using support vector machines". *Machine learning* 46.1-3 (2002), str. 389–422.

[34] T. Hastie, R. Tibshirani i J. Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., 2009.

[35] Hex-Rays. *IDA Pro - Interactive Disassembler*. https://www.hex-rays.com/products/ida/. pristupano 10.9.2014.

[36] F. Hussain. *Microsoft Windows Devices Responsible For 80% of Malware Infections*. https://www.hackread.com/microsoft-windows-devices-malware-infections/. pristupano 1.3.2017.

[37] S. ICS. *Analysis of the Cyber Attack on the Ukrainian Power Grid*. 2016.

[38] I. Institute. *Malware Researcher's Handbook (Demystifying PE File*. http://resources.infosecinstitute.com/2-malware-researchers-handbook-demystifying-pe-file/. pristupano 27.4.2017.

[39] International Secure System Lab. *Anubis - Malware Analysis for Unknown Binaries*. https://anubis.iseclab.org/. pristupano 10.9.2014.

[40] R. Islam i dr. "Classification of malware based on integrated static and dynamic features". *Journal of Network and Computer Applications* 36.2 (2013).

[41] G. James i dr. *An Introduction to Statistical Learning: With Applications in R*. Springer Texts in Statistics. Springer London, Limited, 2013. URL: http://books.google.hr/books?id=at1bmAEACAAJ.

[42] J. Jang. "Scaling Software Security Analysis to Millions of Malicious Programs and Billions of Lines of Code". Disertacija. Carnegie Mellon University, 2013.

[43] J. Jang, D. Brumley i S. Venkataraman. "Bitshred: feature hashing malware for scalable triage and semantic analysis". *Proceedings of the 18th ACM conference on Computer and communications security*. ACM. 2011.

[44] JoeSecurity. *Joe Sandbox: Automated Malware Analysis*. https://www.joesecurity.org/. pristupano 27.8.2017.

[45] G. H. John, R. Kohavi, K. Pfleger i dr. "Irrelevant features and the subset selection problem". *Machine learning: proceedings of the eleventh international conference*. 1994, str. 121–129.

[46] N. Jovanovic, C. Kruegel i E. Kirda. "Pixy: A static analysis tool for detecting web application vulnerabilities". *Security and Privacy, 2006 IEEE Symposium on*. IEEE. 2006.

[47] Jutarnji.hr. *VELIKI NAPAD U TIJEKU! Banda hakera krade i s hrvatskih računa, evo kako se zaštititi*. http://www.jutarnji.hr/banda-hakera-krade-i-s-hrvatskih-racuna-/1196442/. pristupano 15.4.2015.

[48] M. G. Kang, P. Poosankam i H. Yin. "Renovo: A hidden code extractor for packed executables". *Proceedings of the 2007 ACM workshop on Recurring malcode*. ACM. 2007.

[49] F. KernelMode.info. *VBoxAntiVMDetectHardened mitigation*. http://www.kernelmode.info/forum/viewtopic.php?f=11&t=3478. pristupano 21.6.2016.

[50] T. Kišasondi, D. Klasić i Ž. Hutinski. "A multiple layered approach to malware identification and classification problem". *Central European Conference on Information and Intelligent Systems*. 2010.

[51] J. Z. Kolter i M. A. Maloof. "Learning to Detect and Classify Malicious Executables in the Wild". *Journal of Machine Learning Research* 7 (2006).

[52] J. Z. Kolter i M. A. Maloof. "Learning to Detect Malicious Executables in the Wild". *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '04. New York, USA: ACM, 2004.

[53] S. Krasser, B. Meyer i P. Crenshaw. "Valkyrie: Behavioral malware detection using global kernel-level telemetry data". *Machine Learning for Signal Processing (MLSP), 2015 IEEE 25th International Workshop on*. IEEE. 2015, str. 1–6.

[54] C. Kruegel i dr. "Polymorphic worm detection using structural information of executables". *Recent Advances in Intrusion Detection*. Springer. 2006.

[55] D. Kushner. *The Real Story of Stuxnet*. http://spectrum.ieee.org/telecom/security/the-real-story-of-stuxnet. pristupano 14.8.2014.

[56] A. Lanzi i dr. "AccessMiner: Using System-centric Models for Malware Protection". *Proceedings of the 17th ACM Conference on Computer and Communications Security*. CCS '10. New York, USA: ACM, 2010.

[57] R. Lemos. *Stuxnet attack more effective than bombs*. http://www.infoworld.com/article/2625351/malware/stuxnet-attack-more-effective-than-bombs.html. pristupano 15.4.2015.

[58] S. Marsland. *Machine learning: an algorithmic perspective*. CRC press, 2015.

[59] L. Martignoni, M. Christodorescu i S. Jha. "Omniunpack: Fast, generic, and safe unpacking of malware". *Computer Security Applications Conference ACSAC 2007*. IEEE. 2007.

[60] R. A. Maxion i R. R. Roberts. *Proper Use of ROC Curves in Intrusion/Anomaly Detection*. Teh. izv. CS-TR-871. School of Computing Science, University of Newcastle upon Tyne, 2004.

[61] Microsoft. *Peering Inside the PE: A Tour of the Win32 Portable Executable File Format*. https://msdn.microsoft.com/en-us/library/ms809762.aspx. pristupano 27.4.2017.

[62] B. A. Miller. "Scalable platform for malicious content detection integrating machine learning and manual review". (2015).

[63] T. M. Mitchell. *Machine Learning*. 1. izdanje. New York, USA: McGraw-Hill, Inc., 1997.

[64] A. Moser, C. Kruegel i E. Kirda. "Limits of static analysis for malware detection". *Computer Security Applications Conference ACSAC 2007. Twenty-Third Annual*. IEEE. 2007.

[65] J. Newsome i D. Song. "Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software". (2005).

[66] Oracle. *VirtualBox*. https://www.virtualbox.org. pristupano 31.8.2014.

[67] A. Ortega. *Hardening Cuckoo Sandbox against VM aware malware*. https://github.com/a0rtega/pafish. pristupano 21.6.2016.

[68] PcWorld. *Antivirus is dead, says maker of Norton Antivirus*. http://www.pcworld.com/article/2150743/antivirus-is-dead-says-maker-of-norton-antivirus.html. pristupano 15.4.2015.

[69] R. Perdisci, A. Lanzi i W. Lee. "Classification of packed executables for accurate computer virus detection". *Pattern Recognition Letters* 29.14 (2008).

[70] M. Pietrek. *An In-Depth Look into the Win32 Portable Executable File Format*. https://msdn.microsoft.com/en-us/magazine/cc301805.aspx. pristupano 28.3.2015.

[71] Prowling.nu. *Modifying VirtualBox settings for malware analysis*. http://blog.prowling.nu/2012/08/modifying-virtualbox-settings-for.html. pristupano 21.6.2016.

[72] Quick EMUlator. *QEMU - open source process emulator*. http://wiki.qemu.org/Main_Page. pristupano 31.8.2014.

[73] J. Rabaiotti. "Counter Intrusion Software Malware Detection using Structural and Behavioural Features and Machine Learning". (2007).

[74] K. Raman. "Selecting features to classify malware". *InfoSec Southwest* (2012).

[75] S. Raschka. *Python Machine Learning*. Birmingham, UK: Packt Publishing, 2015.

[76] K. Rieck i dr. "Learning and classification of malware behavior". *Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 2008.

[77] P. Royal i dr. "Polyunpack: Automating the hidden-code extraction of unpack-executing malware". *Computer Security Applications Conference ACSAC 2006*. IEEE. 2006.

[78] M. Russinovich. *Process Explorer*. http://technet.microsoft.com/en-us/sysinternals/bb896653.aspx. pristupano 10.9.2014.

[79] M. Russinovich i B. Cogswell. *Process Monitor*. http://technet.microsoft.com/en-us/sysinternals/bb896645.aspx. pristupano 10.9.2014.

[80] Sans. *Sophos detecting itself as SHH/Updater-B*. https://isc.sans.edu/diary/Sophos+detecting+itself+as+SHHUpdater-B/14131. pristupano 21.4.2015.

[81] I. Santos i dr. "Collective Classification for Packed Executable Identification". *Proceedings of the 8th Annual Collaboration, Electronic Messaging, Anti-Abuse and Spam Conference*. CEAS '11. New York, USA: ACM, 2011.

[82] I. Santos i dr. "Opem: A static-dynamic approach for machine-learning-based malware detection". *International Joint Conference CISIS/ICEUTE/SOCO Special Sessions 2012*. Springer. 2013.

[83] J. Saxe i K. Berlin. "Deep neural network based malware detection using two dimensional binary program features". *Malicious and Unwanted Software (MALWARE), 2015 10th International Conference on*. IEEE. 2015, str. 11–20.

[84] M. G. Schultz i dr. "Data mining methods for detection of new malicious executables". *Security and Privacy, 2001. S&P 2001. Proceedings. 2001 IEEE Symposium on*. IEEE. 2001.

[85] D. Sculley i dr. "Detecting Adversarial Advertisements in the Wild". *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD 2011. New York, USA: ACM, 2011.

[86] P. Security. *Hybrid Analysis*. https://www.reverse.it/. pristupano 27.8.2017.

[87] M. Z. Shafiq i dr. "Pe-miner: Mining structural information to detect malicious executables in realtime". *Recent Advances in Intrusion Detection*. Springer. 2009.

[88] M. Sharif i dr. "Eureka: A framework for enabling static malware analysis". *Computer Security-ESORICS 2008*. Springer, 2008.

[89] J. Shlens. "A tutorial on principal component analysis". *arXiv preprint arXiv:1404.1100* (2014).

[90] M. Sikorski i A. Honig. *Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software*. 1st. San Francisco, CA, USA: No Starch Press, 2012.

[91] T. Šmuc. *Strojno učenje, Ansambli - Materijali s predavanja na PMF-MO*. https://web.math.pmf.unizg.hr/nastava/su/index.php/download_file/-/view/112/. pristupano 14.2.2017.

[92] T. Šmuc. *Strojno učenje - Materijali s predavanja na PMF-MO*. https://web.math.pmf.unizg.hr/nastava/su/. pristupano 14.2.2017.

[93] D. Song i dr. "BitBlaze: A New Approach to Computer Security via Binary Analysis". *Proceedings of the 4th International Conference on Information Systems Security*. ICISS '08. Berlin, Heidelberg: Springer-Verlag, 2008.

[94] P. Szor. *The art of computer virus research and defense*. Pearson Education, 2005.

[95] S. M. Tabish, M. Z. Shafiq i M. Farooq. "Malware Detection Using Statistical Analysis of Byte-level File Content". *Proceedings of the ACM SIGKDD Workshop on CyberSecurity and Intelligence Informatics*. CSI-KDD '09. New York, USA: ACM, 2009.

[96] G. Tahan, L. Rokach i Y. Shahar. "Mal-id: Automatic malware detection using common segment analysis and meta-features". *The Journal of Machine Learning Research* 13.1 (2012).

[97] A. Tang, S. Sethumadhavan i S. Stolfo. "Unsupervised Anomaly-based Malware Detection using Hardware Features". *arXiv preprint arXiv:1403.1631* (2014).

[98] theregister.co.uk. *Panda antivirus labels itself as malware, then borks EVERYTHING*. http://www.theregister.co.uk/2015/03/11/panda_antivirus_update_self_pwn/. pristupano 21.4.2015.

[99] K. M. Ting i I. H. Witten. "Issues in stacked generalization". *J. Artif. Intell. Res.(JAIR)* 10 (1999), str. 271–289.

[100] Triumfant. *The Worldwide Malware Signature Counter*. http://www.triumfant.com/Signature_Counter.asp. 2014.

[101] G. Vigna. *Antivirus Isn't Dead, It Just Can't Keep Up*. http://labs.lastline.com/lastline-labs-av-isnt-dead-it-just-cant-keep-up. pristupano 28.8.2014.

[102] C. Willems, T. Holz i F. Freiling. "Toward automated dynamic malware analysis using Cwsandbox". *IEEE Security and Privacy* 5.2 (2007).

[103] Windows Debugger. *WinDbg*. http://www.windbg.org/. pristupano 10.9.2014.

[104] Wired. *Insane 81 milion Bangladesh bank heist here is what we know*. 2016.

[105] D. H. Wolpert. "Stacked generalization". *Neural networks* 5.2 (1992), str. 241–259.

[106] Y. Ye i dr. "An intelligent PE-malware detection system based on association mining". *Journal in Computer Virology* 4.4 (2008). URL: http://dx.doi.org/10.1007/s11416-008-0082-4.

[107] H. Yin i D. Song. "Temu: Binary code analysis via whole-system layered annotative execution". *Submitted to: VEE* 10 (2010).

[108] H. Yin i dr. "Panorama: capturing system-wide information flow for malware detection and analysis". *Proceedings of the 14th ACM conference on Computer and communications security*. ACM. 2007.

[109] O. Yuschuk. *OllyDbg*. http://www.ollydbg.de/. pristupano 10.9.2014.

Životopis

Toni Gržinić rođen je u Puli 19.12.1985. Nakon završene Prirodoslovne matematičke gimnazije u Puli, upisuje Fakultet organizacije i informatike u Varaždinu 2004. godine. Diplomirao je 2010. godine s temom Izbor ERP sustava metodama višekriterijskog odlučivanja. Od 2009. radi u poduzeću Penta d.o.o u Puli na poslovima izrade sustava za javni gradski prijevoz. Godine 2011. prelazi u Hrvatsku akademsku mrežu CARNet u Odjel za računalnu sigurnost i kasnije u Nacionalni CERT. Sudjelovao je na različitim edukacijama i vježbama (NATO CMX, NATO Cyber Coalition, Enisa Cyber Europe) vezanim za računalnu sigurnost. Također, sudjelovao je na europskom projektu ACDC: Advanced Cyber Defence Centre. U akademskoj godini 2011./2012. upisuje doktorski studij Informacijskih znanosti na Fakultetu organizacije i informatike. Trenutno radi u tvrtci Diverto d.o.o kao konzultant za informacijsku sigurnost. Znanstveni interes vezan mu je za područja dubinske analize podataka i računalne sigurnosti. Objavio je 6 znanstvenih radova te redovno sudjeluje i predaje na stručnim konferencijama.

Objavljeni radovi

1. M. Velic, T. Grzinic i I. Padavic. "Wisdom of crowds algorithm for stock market predictions". *35th International Conference on Information Technology Interfaces (ITI), Proceedings of the ITI 2013*. IEEE. 2013, str. 137–144
2. M. Velic, T. Grzinic i M. Drobec. "Feasibility Study on Implementation of Electronic Invoicing in Public Administration Enterprise". *Central European Conference on Information and Intelligent Systems 2013*. Faculty of Organization i Informatics Varazdin. 2013, str. 262
3. T. Grzinic, T. Kisasondi i J. Saban. "Detecting anomalous Web server usage through mining access logs". *24rd Central European Conference on Information and Intelligent Systems*. University of Zagreb, Faculty of Organization i Informatics Varaždin. 2013
4. T. Grzinic i dr. "CROFlux—Passive DNS method for detecting fast-flux domains". *2014 37th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. IEEE. 2014, str. 1376–1380
5. J. Saban, T. Grzinic i L. Mrsic. "An event based framework for facilitating database activity tracking". *Central European Conference on Information and Intelligent Systems 2014*. University of Zagreb, Faculty of Organization i Informatics Varaždin. 2014
6. T. Grzinic, L. Mrsic i J. Saban. "Lino-An Intelligent System for Detecting Malicious Web-Robots". Sv. 9012. Lecture Notes in Computer Science. Springer, 2015, str. 559–568