PROJEKTNI PRIJEDLOG

Klasifikacija zlonamjernog softvera

Matea Stanišić
stmatea@student.math.hr

Petra Škrabo
skpetra@student.math.hr

Mateja Terzanović
matterz@student.math.hr

Margarita Tolja
tomarga@student.math.hr

1 Uvodni opis problema

Jedna od najbolje financiranih i najvažnijih IT industrija današnjice zasigurno je ona koja se bavi zlonamjernim softverima (eng. *malware*). Većina *malware*-a kreira se konstantnim „malicioznim“ mijenjanjem već postojećih, što dovodi do velikog broja različitih datoteka koje predstavljaju istu kategoriju malware-a, tj. imaju isti obrazac malicioznog ponašanja. Ogromna količina podataka koje treba provjeriti predstavlja jedan od najvećih problema s kojima se susreću tvrtke za internetsku sigurnost te stoga na važnost dolazi klasifikacija *malware*-a.

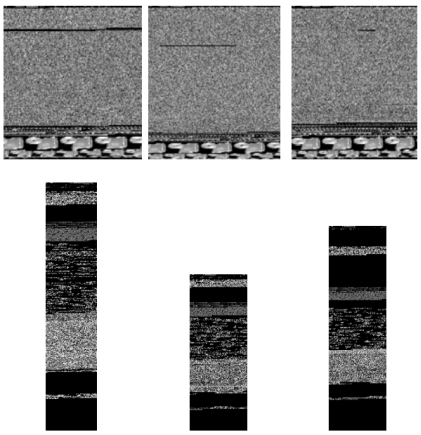
Ovaj projekt je odgovor na *Microsoft Malware Classification Challenge* [7] koji je 2015. godine predstavljen na *Kaggle*-u. Microsoft je objavio skup podataka od oko `0.5 TB` koji sadrži više od 20 000 primjeraka *malware*-a podijeljenih na podatke za treniranje i testiranje. Podaci za treniranje imaju specificiranu jednu od devet klasa kojima mogu pripadati (*Ramnit, Lollipop, Kelihos_ver3, Vundo, Simba, Tracur, Kelihos_ver1, Obfuscator.ACY, Gatak*). Svaki *malware* je identificiran jedinstvenim hash kodom te je reprezentiran dvjema datotekama – heksadecimalnom reprezentacijom binarnog sadržaja (.bytes datoteka) i datotekom koja sadrži metapodatke (funkcijski pozivi, stringovi) izvučene iz binarnih podataka korištenjem *Interactive Disassembler*-a (IDA) (.asm datoteka).

Primjer linije iz jedne .bytes datoteke:
`00401000 56 8D 44 24 08 50 8B F1 E8 1C 1B 00 00 C7 06 08`
Prvi broj, npr. `00401000` označava početak adrese desno napisane heksadecimalne reprezentacije binarnog koda u memoriji. Ostale, desne, heksadecimalne vrijednosti predstavljaju važne informacije o samom *malware*-u, npr. instrukcije u kodu. To se može vidjeti i u primjeru linije iz jedne .asm datoteke:
`.text:00401026 E9 26 1C 00 00 jmp sub\_402C51`

Kako originalni podaci nemaju točno određene značajke koje bi upotrijebili u algoritmu za klasificiranje, rudarili smo podatke po uzoru na članak [1] koristeći njihov gotov kod koji su objavili, uz male promjene.

1.1 Značajke izvučene iz .bytes datoteka

1. **Unigrami** - informacije o frekvencijama svakog od `256 byte`-a
2. **Metapodaci** - veličina .bytes datoteke te adresa prve bit sekvence
3. **Entropija** - mjera količine „nereda“ u distribuciji `byte`-ova
4. **Slikovna reprezentacija *malware*-a** različitih familija - ukoliko svaki bajt reprezentiramo kao količinu sive boje jednog piksela na slici, slijede određeni uzorak. Primjer slikovnih reprezentacija *malware*-a možemo vidjeti na Slici 1.



Slika 1: Primjer slikovne reprezentacije *malware*-a. Gornje tri slike reprezentiraju *malware* iz familije *Fakerean*, dok donje tri iz familije *Dontovo.A*.

5. **Duljine stringova** - broj stringova određenih duljina dobivenih pretvorbom bajtova u pripadajuće ASCII znakove

1.2 Značajke izvučene iz .asm datoteka

1. **Metapodaci** – veličina .asm datoteke i broj linija unutar datoteke
2. **Simboli** `+, -, *, ], [, ?, @` – broj pojavljivanja navedenih simbola
3. ***Operation Code* (OPC)** – broj pojavljivanja instrukcija koje specificiraju koje operacije procesor treba učiniti
4. **Registri** - frekvencije korištenja registara koji nisu namjenjeni za zadatke
5. **Sekcije** – razne karakteristike izvučene iz nekih predefiniranih dijelova od kojih se sastoji svaki PE (*Portable Executable*)
6. **Ključne riječi** - frekvencije korištenja nekih ključnih riječi
7. **API** - Značajke koje predstavljuju frekvencije pozivanja svakog od 794 najkorištenijih API-ja u zlonamjernim datotekama
8. ***Data define*** – razmjer korištenja `db`, `dw` i `dd` instrukcija

U ovom radu se bavimo sustavom temeljenim na učenju koji koristi razne karakteristike *malware*-a kako bi efektivno razvrstao uzorke *malware*-a u pripadajuće skupine. Koristimo se neraspakiranim podatcima unatoč tome što raspakiranje može dovesti do izvlačenja vrijednijih značajki zbog toga što je raspakiravanje izrazito zahtjevan zadatak.

2 Cilj i hipoteze istraživanja problema

Glavni cilj istraživanja je pokušati sa što većom točnošću klasificirati *malware*-e iz podataka za testiranje, odnosno odrediti kojoj od navedenih devet skupina oni pripadaju. Cilj je i analizirati kako najuspješniji model za klasifikaciju vrši klasificiranje, odnosno koje se značajke najviše koriste.

Očekujemo da će se neke značajke pokazati korisnijima te da ćemo moći eliminirati velik broj značajki kako bi pojednostavnili model i time uspostavili ravnotežu između kompleksnosti i performanse.

3 Pregled dosadašnjih istraživanja

Detekcija i klasifikacija *malware*-a su dva zasebna mehanizma koja koriste tvrtke za internetsku sigurnost. Program se prvo treba analizirati da bi se utvrdilo da li sadrži zlonamjeran sadržaj, zatim ako je *malware* pronađen potrebno ga je pridružiti odgovarajućoj kategoriji pomoću klasifikacijskog mehanizma. Ovi sustavi se ugrubo dijele na dva dijela – dinamički i statički.

Dinamički sustavi se bave analizom ponašanja programa za vrijeme izvršavanja, primjerice promatrajući interakcije programa s operacijskim sustavom kroz analizu API poziva. Nedostatci ovakvog pristupa su problematične i nejasne pretpostavke o okolnostima u kojima je stvaran skup podataka, među kojima je i manjak detalja o poduzetim mjerama opreza protiv samih *malware*-a. Također, moderni *malware*-i mogu detektirati da se nalaze u okruženju u kojem se koristi dinamička analiza i u tom slučaju prekinu svoj negativan utjecaj.

Statički sustavi se baziraju na sadržaju i provode analizu *malware*-a bez izvršavanja programa. Najutjecajniji statički pristupi su tzv. SAFE i SAVE koji su predložili analizu raznih obrazaca za otkrivanje zlonamjernog sadržaja. Izvlače se različite karakteristike iz sintakse i semantike programa. Sustavi bazirani na sadržaju su korisni za procjenu je li detektirani *malware* sličan prethodno viđenoj varijanti, bez obavljanja zahtjevnog zadatka raspakiravanja. Nedostatak je što su statički sustavi podložni neispravnim predviđanjima zbog polimorfizma i metamorfizma *malware*-a. Ovaj rad je baziran na statičkim značajkama.

4 Materijali, metodologija i plan istraživanja

4.1 Materijali

Kao izvor podataka koristimo prethodno opisan *dataset* u 1, a program planiramo implementirati u *Jupyter* bilježnici koristeći `Python`-ove biblioteke za strojno učenje.

4.2 Metodologija i plan istraživanja

Posebnu pozornost posvetit ćemo odabiru značajki kako bi poboljšali i pojednostavnili model. Kako bi odredili važnost značajki, plan nam je prvo isprobati unvarijantnu metodu i metodu slučajnih šuma, a po potrebi i neke druge metode. Zatim ćemo konstruirati model i eksperimentirati s različitim skupovima značajki. Prilikom izrade modela nastojat ćemo isprobati što više algoritama, ali se planiramo najviše posvetiti XGBoost algoritmu koji se pokazao jako uspješnim na raznim *Kaggle*-ovim natjecanjima. Također, u cilju poboljšanja modela nastojat ćemo isprobati *Bagging* metodu, odnosno napraviti veći broj modela koristeći bootstrap replike podataka i time smanjiti varijancu greške.

Kao mjeru uspješnosti klasifikacije koristit ćemo točnost i tzv. *logloss*. Točnost ćemo mjeriti kao udio točnih predviđanja, no to obično nije dovoljno za procjenu robusnosti predviđanja te ćemo stoga također koristiti i *logloss*. *Logloss* je klasifikacijska funkcija gubitka koja se često koristi kao evaluacijska mjera u *Kaggle* natjecanjima. Računa prema sljedećoj formuli

$$logloss = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij} log(p_{ij}) \tag{1}$$

gdje je N broj datoteka u skupu za testiranje, M broj oznaka, `log` prirodni logaritam, y_{ij} je 1 ako je zapažanje i u klasi j, 0 inače i p_{ij} je predviđena vjerojatnost da zapažanje i pripada klasi j. Cilj je minimizirati vrijednost te funkcije. Radi jednostavnosti, rješenje ćemo učitati na *Kaggle* koji će nam kao povratnu informaciju dati mjeru uspješnosti.

5 Očekivani rezultati predložnog projekta

Nadamo se dobiti klasifikator koji će što uspješnije na testnom skupu odrediti kojoj od devet kategorija pojedini *malware* iz skupa pripada.

Literatura

[1] Mansour Ahmadi i dr. „Novel Feature Extraction, Selection and Fusion for Effective Malware Family Classification". Ožujak 2016. DOI: 10.1145/2857705.2857713.

[2] *Harlick texture features*. `http://murphylab.web.cmu.edu/publications/boland/boland_node26.html`. Dohvaćeno: 22-04-2020.

[3] *Hrvatska enciklopedija, entropija*. `https://www.enciklopedija.hr/natuknica.aspx?ID=18042`. Dohvaćeno: 22-04-2020.

[4] *Hrvatski mrežni riječnik*. `http://ihjj.hr/mreznik/page/pojmovnik/6/`. Dohvaćeno: 22-04-2020.

[5] *Making Sense of Logarithmic Loss*. `https://datawookie.netlify.app/blog/2015/12/making-sense-of-logarithmic-loss/`. Dohvaćeno: 22-04-2020.

[6] *Microsoft malware classification Challenge*. `http://arxiv.org/abs/1802.10135`. Dohvaćeno: 13-04-2020.

[7] *Microsoft malware classification Kaggle Challenge*. `https://www.kaggle.com/c/malware-classification`. Dohvaćeno: 13-04-2020.

[8] Lakshmanan Nataraj i dr. „Malware Images: Visualization and Automatic Classification". (Srpanj 2011). DOI: 10.1145/2016904.2016908.

[9] *Tomislav Lipić i Matija Piškorec, Vježbe kolegija Strojno učenje*. https://github.com/pmf-strojnoucenje/Vjezbe. Dohvaćeno: 22-04-2020.

[10] *Tomislav Šmuc, Predavanja iz kolegija Strojno učenje*. Dohvaćeno: 22-04-2020.

[11] *Understanding and Detecting Malware Threats Based on File Size*. https://d3gpjj9d20n0p3.cloudfront.net/fortiguard/research/DetectingMalwareThreats.pdf. Dohvaćeno: 22-04-2020.

[12] *What Can N-Grams Learn for Malware Detection?* https://www.edwardraff.com/publications/what_can_ngrams_learn.pdf. Dohvaćeno: 22-04-2020.