Meta-heuristički algoritmi za otkrivanje zlonamjernog softvera

Stanišić, Matea

Meta-heuristički algoritmi za otkrivanje zlonamjernog softvera

Stanišić, Matea

SVEUČILIŠTE U ZAGREBU

PRIRODOSLOVNO–MATEMATIČKI FAKULTET

MATEMATIČKI ODSJEK

Matea Stanišić

META-HEURISTIČKI ALGORITMI ZA OTKRIVANJE ZLONAMJERNOG SOFTVERA

Diplomski rad

Voditelj rada:
doc. dr. sc. Goranka Nogo

Zagreb, 2020.

Ovaj diplomski rad obranjen je dana _____ pred ispitnim povjerenstvom u sastavu:

1. _____, predsjednik
2. _____, član
3. _____, član

Povjerenstvo je rad ocijenilo ocjenom _____.

Potpisi članova povjerenstva:

1. _____
2. _____
3. _____

Sadržaj

Uvod

Danas sa sigurnošću možemo reći da se čovječanstvo u potpunosti oslanja na računala i druge pametne uređaje zbog čega o njima i ovisi. Nažalost, to dovodi do toga da svaka prijetnja tim uređajima predstavlja prijetnju i društvu. Četiri jahača elektroničke apokalipse, kako ih naziva Aycock u [9], su neželjena pošta, softverski *bugovi*, napadi uskraćivanjem resursa i zlonamjerni softveri. Kako napadi zlonamjernim softverima uzrokuju velike financijske štete, upravo su oni fokus ovoga rada.

Prvenstveno, ovaj rad pružit će pregršt informacija o zlonamjernim softverima među kojima su njegova definicija, kategorije, ali i razvoj kroz povijest. Kratko će se komentirati zaštita od njih te tehnike njegovog otkrivanja što je i cilj ovog rada. Kako bi u tome bili što uspješniji, koriste se meta-heuristike čiji zadatak je pretražiti prostor najboljih modela koji razlikuju zlonamjerne softvere od običnih. Odabrani meta-heuristički algoritam u ovom radu je pčelinje programiranje.

Pregled rada

U Poglavlju 1 upoznati ćemo se s terminom zlonamjernog softvera, njegovim najpoznatijim vrstama i pregledom njegovog razvoja kroz povijest. U Poglavlju 2 predstavljeni su koraci zaštite od zlonamjernog softvera, tehnike njegovog otkrivanja i načini na koje se analizira. Poglavlje 3 opisuje meta-heuristike korištene za rješavanje problema razlikovanja zlonamjernog softvera od običnog. U Poglavlju 4 slijedi opis implementacije algoritma, pregled korisničkog sučelja, opis korištenih podataka te rezultati testiranja. Konačno, slijede zaključak, korištena bibliografija i sažetak.

Poglavlje 1

Zlonamjerni softver

Zlonamjerni softver (eng. *malware*, skraćeno od *malicious software*) je program tajno umetnut u drugi program s namjerom uništavanja podataka, pokretanja razarajućih ili nametljivih programa bez znanja njegovog vlasnika ili bilo kojeg drugog načina ugrožavanja povjerljivosti, cjelovitosti ili dostupnosti žrtvinih podataka, aplikacija ili operacijskog sustava [57]. Može poprimiti oblik skripte, koda, izvršne datoteke ili bilo kojeg drugog programa. Najčešći način prijenosa zlonamjernog softvera s jednog kontaminiranog uređaja na drugi su komunikacijski kanali (elektronička pošta, internet) te prijenosni memorijski uređaji.

Termin zlonamjernog softvera objedinjuje široki spektar softvera koji napadači koriste za:

- špijuniranje korisnika
- preuzimanje kontrole nad uređajima
- krađu osjetljivih (osobnih, poslovnih, financijskih) informacija
- ometanje normalnog rada uređaja
- slanje neželjene pošte
- sudjelovanje u distribuiranim napadima uskraćivanjem resursa (eng. *DDoS*, skraćeno od *distributed Denial of Service attack*)[1]
- zaključavanje datoteka na uređajima te zadržavanje istih radi otkupnine.

Ovisno o funkcionalnosti za koju je specifični zlonamjerni softver namijenjen postoji nekoliko vrsta softvera.

[1] *DDoS* napadi nastaju kada više (prethodno) kompromitiranih sustava poplavljuje resurse ciljanih sustava, obično jednog ili više web poslužitelja [14].

1.1 Vrste zlonamjernog softvera

Kategorizacija zlonamjernih programa može se raditi na različite načine. Primjerice, mogu se grupirati s obzirom za koji operacijski sustav su namijenjeni. No, kako je većina njih namijenjena za napade na uređaje koji koriste operacijski sustav *Windows* [59], ova jednostavna klasifikacija u većini slučajeva ne bi dala mnogo informacija. Također, zlonamjerni programi mogu se razlikovati i po načinu prijenosa ili okruženja u kojem se aktiviraju. Ipak, najčešća i najpoznatija je kategorizacija koja se radi razlučivanjem načina njihovog rada koristeći sljedeće tri osobine:

1. **mogućnost samorepliciranja**
   Zlonamjerni softver koji se *aktivno* pokušava proširiti stvarajući nove instance samog sebe ima mogućnost samorepliciranja, dok onaj koji se širi *pasivno* nema.
2. **rast populacije**
   Ovom osobinom označava se cjelokupni broj instanci zlonamjernog softvera nastalih samorepliciranjem. Ukoliko zlonamjerni softver nema svojstvo samorepliciranja njegov rast populacije uvijek će biti nula. Obrat ne vrijedi, tj. za zlonamjerni softver koji za rast populacije ima nulu nije nužno da nema svojstvo samorepliciranja.
3. **parazitnost**
   Osobina zlonamjernog softvera koji za svoje postojanje zahtijeva postojanje nekog drugog izvršnog programa.

1.1.1 **Virus** (eng. *virus*)

- **Mogućnost samorepliciranja**: Da.
- **Rast populacije**: Pozitivan broj.
- **Parazitnost**: Da.

Virus je termin koji se često poistovjećuje s terminom zlonamjernog softvera jer su se od svih vrsta upravo oni prvi pojavili [59]. To je naravno krivo jer je virus samo jedan od tipova zlonamjernog softvera prepoznatljiv po tome da se nakon pokretanja, bez dopuštenja ili znanja korisnika, pokušava replicirati u druge izvršne programe. Kada u tome uspije, kod tih programa postaje *zarazan*. Pokretanjem zaraženog koda može se zaraziti kod nekog drugog izvršnog programa ponavljanjem početnog postupka repliciranja.

Virusi se najčešće šire s jednog računala na drugo u obliku izvršnog zlonamjernog koda putem interneta, privitaka u elektroničkoj pošti ili raznim memorijskim uređajima.

Antivirusni softver ili antivirus posebna je vrsta programa dizajnirana za zaštitu, otkrivanje i uklanjanje virusa. Iako pojam zlonamjernog softvera danas objedinjuje puno širi spektar nego u prošlosti, termin antivirus danas se koristi i za programe koji štite uređaje od virusa ali i od ostalih vrsta zlonamjernog softvera.

1.1.1.1 Anatomija virusa

Svaki virus sastoji se od tri ključna dijela.

- **Princip zaraze** (eng. *infection mechanism*) je način na koji će se virus replicirati na druge programe. Točno sredstvo kojim se virus širi naziva se njegovim *vektorom zaraze*. On ne mora biti jedinstven, tj. postoje virusi koji se mogu samoreplicirati na druge programe na više različitih načina (eng. *multipartite*).
- **Korisni teret** (eng. *payload*) virusa su podaci koji prenose njegovu stvarnu zlonamjernu namjenu.
- **Okidač** (eng. *trigger*) je način odlučivanja hoće li se korisni teret aktivirati ili ne.

Princip zaraze, korisni teret i okidač zajedno čine *tijelo* virusa. Struktura svakog virusa može se prikazati pseudokodom prikazanim u Pseudokodu 1.1.

Pseudokod 1.1: Struktura virusa [9].

```
def virus():
    infect()
    ako trigger() onda payload()
```

1.1.1.2 Životni ciklus virusa

Postojanje virusa na uređaju može se uočiti u raznim stanjima njegovog djelovanja.

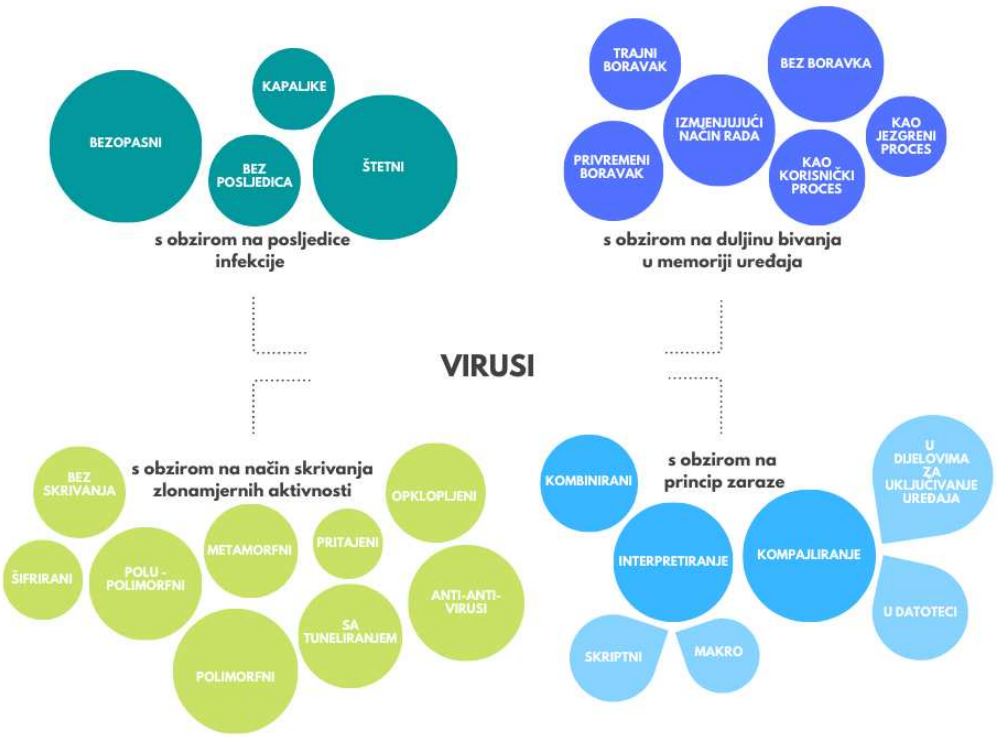
1. **Faza latencije** (eng. *dormant phase*) je stanje u kojemu virus ima pristup uređaju ali još uvijek ne poduzima nikakve akcije. Nije nužno da svi virusi imaju ovo stanje. Izvorni oblik virusa, prije samorepliciranja, naziva se *klica*.
2. **Faza repliciranja** (eng. *propagation phase*) je faza djelovanja u kojemu virus započinje sa samorepliciranjem u druge izvršne programe na uređaju koji tada sadržavaju *klon* klice.
3. **Faza okidača** (eng. *triggering phase*) aktivira se okidačem. On se može pokrenuti raznim sistemskim događajima, npr. određenim brojkom rasta populacije virusa.
4. **Faza izvršavanja** (eng. *execution phase*) je stanje virusnog programa u kojemu se izvršava korisni teret na uređaju.

1.1.1.3 Vrste virusa

Virusi se dodatno mogu grupirati na razne načine. Podjela je prikazana Slikom 1.1.

Jedan od načina klasificiranja virusa je **s obzirom na posljedice njegove infekcije** (eng. *payload based*). Koristeći ovu karakteristiku, postoje četiri kategorije virusa.

1. **Virusi bez posljedica** (eng. *no payload*), odnosno oni bez osnovnog tereta, su svi oni koji nemaju nikakvu drugu ulogu osim one da se repliciraju i šire na druge programe.



Slika 1.1: Klasifikacija virusa.

2. **Bezopasni virusi** (eng. *non-destructive payload*) u osnovnom teretu najčešće nose određenu poruku ili sliku. Osnovni cilj im je zadirkivati korisnike te time smanjiti njihovu produktivnost.
3. **Štetni virusi** (eng. *destructive*) su, upravo zbog njihovih razornih posljedica, najčešća kategorija virusa korištena u napadima. Posljedice se razlikuju ovisno o virusu.
4. **Kapaljka** (eng. *droppers*) je kategorija koja objedinjuje sve viruse specifične po tome što izvršavanjem svog osnovnog tereta pomažu napadačima u prikupljanju sredstava za izvršavanje nekih drugih zlonamjernih aktivnosti.

S obzirom na duljinu bivanja virusa u memoriji uređaja (eng. *memory based*) razlikuju se:

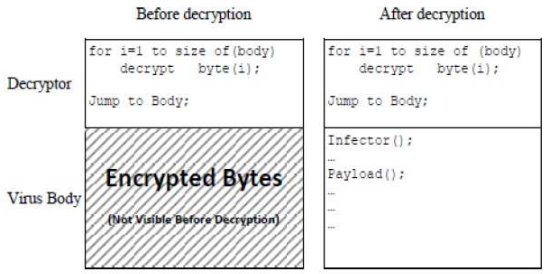
1. Virusi koji **trajno borave** (eng. *resident*) u memoriji uređaja nastoje zaraziti sve bitne datoteke koje postoje u memoriji.
2. Virusi koji **privremeno borave** (eng. *temporary resident*) u memoriji uređaja ostaju

tamo sve dok se ne dogodi neki prethodno definirani sistemski događaj. Kako je virus aktivan samo u određenom vremenskom periodu između događaja, teško ga je otkriti.

3. Virusi s **izmjenjujućim načinom rada** (eng. *swapping mode*) u memoriji uređaja predstavljaju kombinaciju prethodne dvije kategorije. Dakle, to su svi virusi koji pohrane samo dio svog koda u memoriju uređaja, i to samo u vremenskom periodu između dva prethodno definirana sistemska događaja.
4. Virusi koji **ne borave** (eng. *non-resident*) u fizičkoj memoriji uređaja, kako bi pronašli druge programe koje bi mogli zaraziti, koriste mehanizme za pretraživanje i kopiranje.
5. Virusi koji se **pokreću kao korisnički proces** (eng. *user process*) repliciraju se na sve datoteke koje su im dostupne.
6. Virusi koji se **pokreću kao jezgreni proces** (eng. *kernel process*) imaju privilegije nakon što zaraze uređaj jer se nalaze u njegovoj jezgri. Na jezgru se uglavnom uspiju prikačiti kroz upravljačke programe. Kako ovi virusi vrše izmjene u glavnom datotečnom sistemu, za njihovo izvršavanje potrebne su dozvole administratora.

Ambicija svakog tvorca virusa zasigurno je kreirati virus koji se neće moći otkriti niti analizirati. Kako bi u tome uspjeli, služe se **tehnikama za skrivanje zlonamjernih aktivnosti** (eng. *obfuscation techniques*). Obzirom na tehniku koju koriste, moguće je razlikovati devet vrsta virusa.

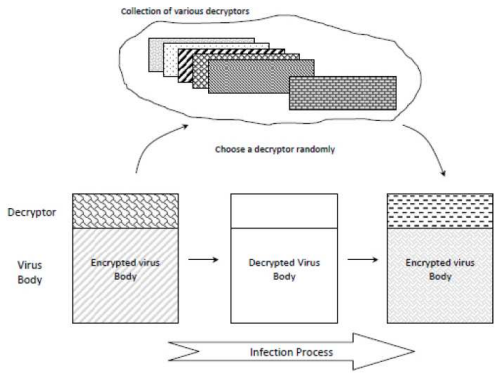
1. Virusi **bez tehnika za skrivanje** (eng. *no obfuscation*) su najjednostavniji za kreirati, ali i za otkriti i analizirati.
2. Virusi koji koriste **šifriranje** (eng. *encryption*) kako bi prekrili svoju zlonamjernu funkcionalnost sastoje se od dva dijela - šifriranog tijela virusa i petlje koja ga dešifrira. Njihova struktura se može vidjeti na Slici 1.2.



Slika 1.2: Struktura šifriranog virusa [11].

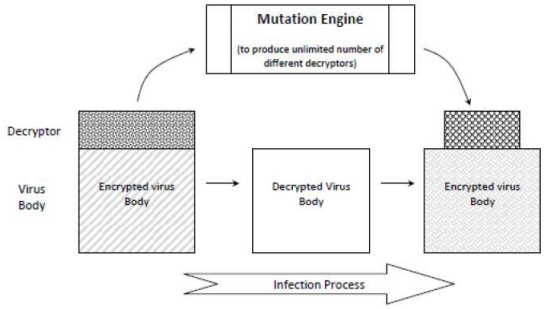
3. **Polu-polimorfni** (eng. *oligomorphism*) virusi, uz pretpostavku da je ključ svakog šifriranog klona nasumičan, služe se različitim tehnikama dešifriranja tijela kako bi

izbjegli otkrivanje čitanjem njihovih potpisa. Tehnika za dešifriranje ima konačno mnogo. Struktura i mehanizam rada ove grupe virusa prikazani su na Slici 1.3.



Slika 1.3: Struktura i mehanizam rada polu-polimorfnog virusa [11].

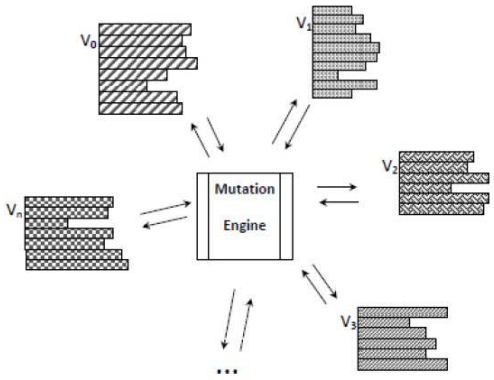
4. **Polimorfni** (eng. *polymorphism*) virusi temelje se na izmjeni izgleda koda[2] petlje za dešifriranje svakog klona koja se postiže mijenjanjem tehnike za dešifriranje kojih ima gotovo beskonačno. Struktura i mehanizam rada ove grupe virusa može se vidjeti na Slici 1.4.



Slika 1.4: Struktura i mehanizam rada polimorfnog virusa [11].

5. **Metamorfni** (eng. *metamorphism*) virusi, umjesto šifriranja, koriste se metodama izmjene koda tijela napravom za mutacije pri svakoj replikaciji. Pri tome, funkcionalnost klonova ostaje ista. Prikaz širenja metamorfnog virusa može se vidjeti na Slici 1.5.
6. **Pritajeni** (eng. *stealth*) virusi, za razliku od prethodnih kategorija koji skrivaju samo tijelo virusa, pokušavaju spriječiti njihovo otkrivanje aktivnim prikrivanjem

[2]Izmjena koda vrši se pomoću naprave za mutacije (eng. *mutation engine*) koja se sastoji od skupa ekvivalentnih isječaka koda koje koristi kako bi izmjenila ulazni programski kod s nekim njemu ekvivalentnim.



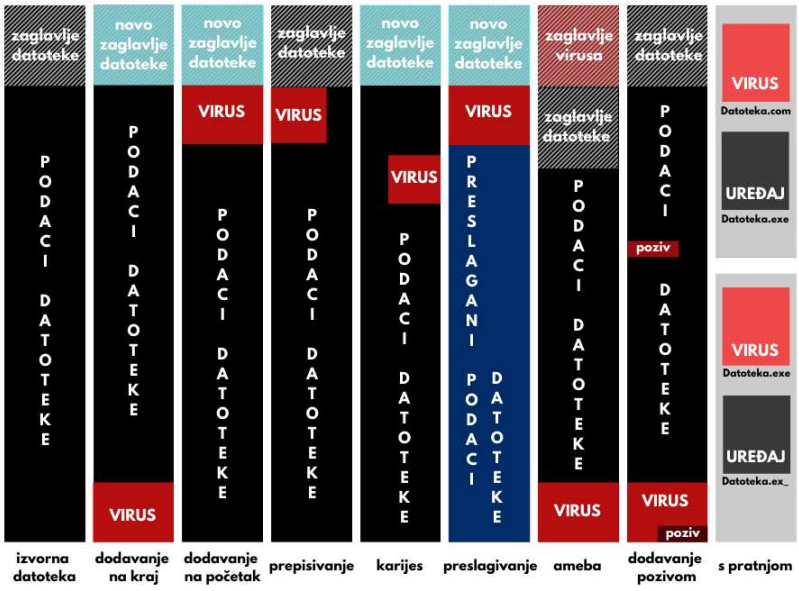
Slika 1.5: Prikaz širenja metamorfnog virusa [11].

samorepliciranja, odnosno zaraze. Tehnikama kao što su povratak na postavke uređaja prije zaraze ovi virusi pokušavaju biti neprimjetni svim programima, a ne samo programima namijenjenim za njihovo otkrivanje.

7. **Oklopljeni** (eng. *armoring*) virusi su svi oni čiji je osnovni cilj, jednom kada uređaj prepozna njihove zlonamjerne aktivnosti, zakomplicirati njihovu analizu. Primjer jedne takve tehnike je mogućnost prepoznavanja da se radi o virtualnoj okolini.
8. Virusi koji se koriste **tuneliranjem** (eng. *tunneling*) kako bi spriječili njihovo otkrivanje izvršavaju se nakon sistemskih prekida.
9. **Anti-antivirusi** su svi virusi koji ili agresivno napadaju antivirusne softvere ili raznim tehnikama pokušavaju antivirusima otežati njihovu analizu ili se pokušavaju spriječiti svoje otkrivanje koristeći znanje kako antivirusni softver radi.

Viruse možemo klasificirati i **s obzirom na princip zaraze** (eng. *target based*) u sljedeće tri kategorije:

1. **Kompajlirani** (eng. *compiled*) virusi su svi oni čiji je kod kompajliran u izvršnu datoteku kako bi ga mogao pokrenuti operacijski sustav. Ovi virusi mogu se dodatno grupirati u viruse koji inficiraju datoteke na uređaju (eng. *file infector*) i na one koji inficiraju dijelove za uključivanje uređaja (eng. *boot sector*). Na Slici 1.6 prikazani su načini na koje virusi mogu inficirati datoteke na uređaju.
2. **Interpretirani** (eng. *interpreted*) virusi su svi oni čiji je kod interpretiran nekom aplikacijom. Dodatno se razlikuju po tome koriste li makroe ili skripte kako bi se proširili na druge programe.
3. Virusi koji koriste **više od jednog principa zaraze** (eng. *multipartite*).



Slika 1.6: Različiti načini na koji virusi mogu inficirati datoteke na uređaju [25].

1.1.2 Crv (eng. *worm*)

- **Mogućnost samorepliciranja**: Da.
- **Rast populacije**: Pozitivan broj.
- **Parazitnost**: Ne.

Crv je vrsta zlonamjernog programa vrlo sličan virusu jer sam sebe umnožava. Struktura im je gotovo ista onoj u Pseudokodu 1.1, ali s vrlo bitnom razlikom - crvi se šire isključivo računalnim mrežama. Također, za razliku od virusa, crvi ne zahtijevaju postojanje domaćinske datoteke za svoj rad. Oni su samostalni programi koji se u većini slučajeva šire bez interakcije korisnika. Iako je moguće pronaći računalne crve koji nisu štetni, većina sigurnosnih stručnjaka sve crve smatra zlonamjernim i nepoželjnim programima [15].

Postoje dvije vrste crva:

- **Crvi mrežne usluge** (eng. *network service worms*) iskorištavaju ranjivost mrežnih usluga kako bi se razmnožili i zarazili druge uređaje i servere. Kako ovi crvi djeluju u potpunosti bez ljudskih aktivnosti, uobičajeno se prošire brže od bilo koje druge vrste zlonamjernog softvera.

- **Crvi masovnih emailova** (eng. *mass mailing worms*) su vrsta crva koja dijeli sličnosti s virusima nošenim električnom poštom. Međutim, ključna razlika je da crvi ne zahtijevaju postojanje domaćinske datoteke za svoj rad. Jednom kada ovaj crv zarazi uređaj, uobičajeno pretražuje adresar elektroničke pošte korisnika uređaja kako bi se dalje propagirao slanjem kopija samoga sebe pronađenim adresama.

Zec (eng. *rabbit*) je vrsta crva specifična po tome da „skače" s jednog uređaja na drugi, umjesto klasičnoga samorepliciranja kopiranjem. Odnosno, rast populacije zeca je nula.

Termin „crv" prvi puta je upotrijebio John Brunner 1975. godine u svom znanstvenofantastičnom romanu *The Shockwave Rider* gdje je opisao program koji se širi putem računalne mreže. Prva značajna izvedba takvog programa dogodila se 1988. godine. Program je nazvan „Crv Moriss" po njegovom tvorcu Robertu Morrisu Jr. koji je, u pokušaju prebrojavanja svih uređaja spojenih na internet, uspio onesposobiti isti stvaranjem prevelikog mrežnog prometa. Zbog toga se često naziva i „Internetski crv".

1.1.3 Logička bomba (eng. *logic bomb*)

- **Mogućnost samorepliciranja**: Ne.
- **Rast populacije**: Nula.
- **Parazitnost**: Moguće.

Logička bomba je vrsta zlonamjernog softvera prepoznatljiva po izvršavanju skupa instrukcija koje ugrožavaju sustave koristeći logiku koju je definirao njihov tvorac.

Svaka logička bomba sastoji se od dva dijela:

1. **Korisni teret** (eng. *payload*) je bilo kakva zlonamjerna operacija koja se izvršava.
2. **Okidač** (eng. *trigger*) je način odlučivanja hoće li se korisni teret aktivirati ili ne. Može se definirati na razne načine (npr. koristeći određeni vremenski period, sistemske događaje ili verziju operacijskog sustava) te se na isti ili drugačiji način može i isključiti.

Logičke bombe mogu biti dio samostalnog programskog koda ili umetnute u već postojeći. Primjer jednostavne logičke bombe može se vidjeti u Pseudokodu 1.2.

Pseudokod 1.2: Jednostavna logička bomba [9].

postojeći programski kod
ako `date is Friday the 13th` **onda** `crash_computer()`
postojeći programski kod

Logičke bombe mogu biti suptilne, posebice u velikim projektima s ograničenim provjerama koda. Napadači su najčešće nezadovoljni zaposlenici koji na taj način kasnije ucjenjuju poslodavca. Dobar primjer logičke bombe predstavlja incident kada je u listopadu 2005. godine Mark Russinovich otkrio da je *Sony BMG* u svoje glazbene CD-ove ugradio logičku bombu koja je prikriveno i bez pristanka instalirala nesigurni softver na korisničkom uređaju [48]. Instalirani zlonamjerni softver pratio je i izvještavao o navikama slušanja kupaca te promijenio pristup operacijskog sustava njihovoj strojnoj opremi. Nesigurni udaljeni pristupi koji su se izgradili instalacijom tog softvera omogućili su pristup i drugim zlonamjernim programima. *Sony* je prodao otprilike 22 milijuna takvih CD-ova.

Zlonamjerni napadi koji koriste logičke bombe mogu se ublažiti korištenjem statičke i dinamičke analize programskog koda, pažljivom analizom istoga te temeljitom verifikacijom i validacijom softverskih paketa.

1.1.4 Trojanski konj (eng. *trojan horse*)

- **Mogućnost samorepliciranja**: Ne.
- **Rast populacije**: Nula.
- **Parazitnost**: Da.

Termin „trojanski konj" dolazi iz grčke mitologije. Nakon 10 godina ratovanja s Trojom, Grci su podmuklo probili njihove zidove. U tome su uspjeli izgradnjom velikog, šupljog drvenog konja u kojeg su smjestili svoje ratnike kako bi im kasnije, nakon što su Trojanci prihvatili skulpturu kao poklon u znak mira, otvorili vrata grada. Slično značenje ima isti termin u računarstvu. Trojanski konj, ili kratko - trojanac, je svaki program koji je naizgled benigan, ali zapravo potajno, u pozadini, bez znanja korisnika, izvršava neki drugi, zlonamjerni pothvat. Trojanci su parazitni programi koji se ne repliciraju. Najčešće, jednom kada zaraze uređaj, mijenjaju postojeće datoteke zlonamjernim verzijama ili dodaju nove, šire druge zlonamjerne softvere i remete performanse računala. Nerijetko se koriste i za krađu osobnih podataka upotrebom „hvatača zaporki"[3].

Trojanci se šire preuzimanjem zaraznog softvera, kao email privitci, preko ranjivosti softvera i putem zlonamjernih web stranica s dinamičkim sadržajem. Antivirusni i drugi programi za zaštitu od zlonamjernih programa pružaju zaštitu od trojanskih konja, no ukoliko napadač već ima pristup uređaju, uklanjanje je teško jer je potrebno otkriti sve promjene na uređaju koje je napadač napravio. Tada se uklanjanju zaraze obično pristupa formatiranjem tvrdog diska i reinstalacijom operacijskog sustava.

[3]Programi i aplikacije s naizgled autentičnom formom za unos podataka. Korisnici u nju upisuju svoje korisničko ime i zaporku koje trojanac pohrani, preusmjeri korisnika na ispravnu formu i ispiše poruku o „nesipravnoj zaporci" kako bi korisnik mislio da se radi o njegovoj greški pri upisu znakova s tipkovnice.

Postoje dvije vrste trojanaca:

- Stopostotni trojanski kod koje je jednostavno analizirati.
- Pažljiva dorada orginalne aplikacije s dodatnim funkcionalnostima nastale upotrebom neke druge vrste zlonamjernog programa, najčešće Stražnja vrata.

Sigurnosni stručnjaci koristili su termin „trojanski konj" kakav nam je poznat danas od 1972. godine kada ga je iskoristio James Anderson u dodatku izvješća [6] za američke zračne snage. Već 1975. godine nastao je prvi program koji se ponaša kao trojanski konj - jednostavna igra *ANIMAL*. Kako igra nije imala štetne posljedice, prvim pravim trojanskim konjem smatra se program *AIDS* iz 1989. godine.

1.1.5 Stražnja vrata (eng. *back door*)

- **Mogućnost samorepliciranja**: Ne.
- **Rast populacije**: Nula.
- **Parazitnost**: Moguće.

Mehanizam stražnjih vrata je vrsta zlonamjernog softvera pomoću kojega napadači dobivaju neovlašten, zamjenski pristup uređajima i njihovim podacima zaobilaskom postojećih sigurnosnih mehanizama ugrađenih u sustav. Tvorci stražnjih vrata najčešće su programeri koji ih slučajno kreiraju radi pokretanja dijagnostike u svrhu otklanjanja kvarova, a onda ih zabunom ne uklone. Nerijetko, ovim mehanizmom koriste se i napadači, kako bi imali neometani pristup željenim podacima.

Stražnja vrata jedan su od najopasnijih tipova parazita jer napadaču daju mogućnost provođenja bilo kakvih operacija i procesa na napadnutom uređaju. Koristi se za špijuniranje korisnika, upravljanje njihovim datotekama, kontroliranje cijelog uređaja, napade na druge korisnike i instaliranje dodatnih softvera.

Slično kao i logičke bombe, stražnja vrata mogu biti dio samostalnog programskog koda ili umetnuta u već postojeći. Također, kreiranje stražnjih vrata može se ublažiti temeljitom validacijom i verifikacijom programskog koda. To se pokazalo točnim 2003. godine kada je netko izmijenio dvije linije u izvornom kodu *Linux* jezgre nevidljive ljudskom oku, ali prepoznate pomoću automatske analize koda. Izmjene koje je nepoznati napadač napravio omogućile bi napadačima ulogu administratora na računalima s *Linux* operacijskim sustavom. Jednostavniji primjer umetnutih stražnjih vrata može se vidjeti u Pseudokodu 1.3.

Pseudokod 1.3: Jednostavna stražnja vrata [9].

```
username = read_username()
password = read_password()
ako username == "hacker" onda
```

```
        return ALLOW_LOGIN
ako valid(username) i valid(password) onda
        return ALLOW_LOGIN
inače return DENY_LOGIN
```

Alat za daljinsko upravljanje (eng. *remote administration tool* - RAT), odnosno trojanski udaljeni pristup (eng. *remote access trojan*), je posebna vrsta stražnjih vrata. Alat izvorno najčešće koriste zaposlenici programske podrške kako bi mogli daljinski dijagnosticirati greške na uređaju i popraviti iste. No, ukoliko RAT na uređaj instalira neki od zlonamjernih softvera, alat postaje štetan „otvaranjem stražnjih vrata" uređaja brojnim napadačima.

Uz zaposlenike programske podrške, alat za daljinsko upravljanje često koriste i zaposlenici mnogih drugih kompanija kako bi mogli pristupiti radnom računalu od kuće. Takav način rada posebno se pokazao korisnim ove godine što je potaklo napadače na povećani broj napada [37], [36].

1.1.6 Špijunski softver (eng. *spyware*)

- **Mogućnost samorepliciranja**: Ne.
- **Rast populacije**: Nula.
- **Parazitnost**: Ne.

Špijunski softver je vrsta zlonamjernog programa specifičan po prikupljanju informacija bez znanja korisnika s uređaja i slanja istih drugome. Ovi zlonamjerni programi nisu parazitni niti samorepli cirajući što ih razlikuje od većine do sada opisanih vrsta.

Vrsta podataka koje špijunski program prikuplja ovisi o vrsti njega samog, ali uobičajeno se radi o:

- **Korisničkom imenu i zaporci** koji se mogu pronaći u datotekama na uređaju ili prikrivenim snimanjem unosa na tipkovnici (eng. *keylogger*) čija je funkcionalnost u zadnje vrijeme narušena zbog sve učestalijeg korištenja dvostruke autentifikacije.
- **Adresama elektroničke pošte** za potrebe kreiranja velikog broja neželjene pošte.
- **Bankovnim računima i podacima o kreditnim karticama** za financijske prevare i krađe identiteta.
- **Licencama za softvere** za internetsko piratstvo.

Špijunski programi do uređaja dolaze na razne načine - posjetom ilegalnih internetskih stranica, posjetom stranica na kojima su napadači iskoristili tehničke nedostatke u internetskim preglednicima (eng. *dive-by download*), zajedno sa softverom kojeg korisnik instalira, a ponekad čak i legitimnog softvera čiji su autori na to potplaćeni.

Prije nego što su ovakvi programi postali sigurnosna prijetnja, termin „špijunski softver" korišten je prvi puta 1995. godine kroz šalu na *Usenet*[4] objavi s ciljem ismijavanja *Microsofovog* poslovnog modela. Američki internetski portal (AOL, skraćeno od eng. *America online*) i državni kibernetički sigurnosni savez 2004. godine proveli su anketu sa zastrašujućim rezultatima. Pokazalo se da 80% internetskih korisnika na svom uređaju imaju instaliran špijunski softver, od kojih 89% njih nisu toga svjesni. Također, od svih korisnika čiji uređaji su zaraženi špijunskim softverom, 95% njih izjavilo je da nisu nikada dopustili da se oni instaliraju.

Neke zemlje, poput Švicarske, Njemačke i Amerike imaju legalni sistem kojim reguliraju upotrebu špijunskog softvera kojeg koristi vlada.

Jedna specifična grupa špijunskog softvera je zlonamjerni oglasni softver (eng. *adware*) koji je prepoznatljiv po koncentriranosti na marketing. Najčešće se pojavljuje kao reklama koja preusmjerava korisnika na određene internetske stranice s ciljem prodaje. U posljednje vrijeme oglašavački softveri koriste se kontekstnim filtriranjem[5] kako bi ponudili korisniku proizvode koji ga zanimaju i time povećali vjerojatnost izvršavanje prodaje.

1.1.7 Ucjenjivački softver (eng. *ransomware*)

- **Mogućnost samorepliciranja**: Da.
- **Rast populacije**: Pozitivan broj.
- **Parazitnost**: Moguće.

Ucjenjivački softver je svaki zlonamjerni program koji korisniku onemogućuje normalno korištenje računala kriptiranjem njegovih dokumenata ili drugih vrijednih podataka. Ova vrsta zlonamjernog softvera razlikuje se od svih drugih jer, nakon što zarazi uređaj i korisniku blokira pristup njegovim podacima, o istome ga obavještava s određenom porukom koju nije moguće maknuti. Cilj ove vrste napada je prisiliti korisnika da plati otkupninu u zamjenu za daljnje normalno korištenje računala i vraćanje pristupa njegovim podacima. S obzirom na ozbiljnost napada, zastrašujuća programska podrška (eng. *scareware*) je najtrivijalniji oblik ucjenjivačkog softvera. Širi se zastrašivanjem korisnika lažnim izjavama (npr. da mu je uređaj zaražen virusom) kako bi ga naveo na instalaciju nekog zlonamjernog programa. Nešto opasnije je zlonamjerno zaključavanje zaslona (eng. *screen lockers*) kod kojega napadač onemogući korisnički uređaj u potpunosti ispisivajući poruke pri pokretanju uređaja o otkupnini. Ucjenjivački programi koji prilikom zaraze uređaja pretražuju

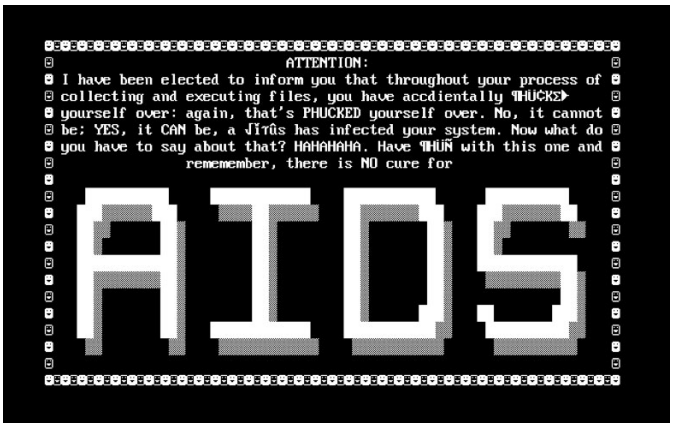
[4] *Usenet* je podmreža na internetu koja se bazira na NNTP protokolu, a čine ju poslužitelji namijenjeni primanju, slanju i razmjenjivanju članaka.

[5] Kontekstno filtriranje (eng. *content-based recommendations*) je kategoriziranje korisničkih podataka na temelju konteksta.

podatke vrijedne korisniku i iste šifriraju pripadaju u najopasniju kategoriju (eng. *encrypting ransomware*).

Ucjenjivački softveri do uređaja uglavnom dolaze putem elektroničke neželjene pošte i iskorištavajući ranjivost mrežnih servisa, a šire se slično kao i crvi. Promatrajući metode koje napadači koriste kako bi proširili zarazu, postoje tri vrste ucjenjivačkog softvera.

- **Kriptocrv** (eng. *cryptoworm*) je samostalan ucjenjivački program kojemu je cilj replicirati se na druge uređaje što većom brzinom i utjecajem.
- **Ucjenjivački softver kao usluga** (eng. *ransomware-as-a-service*, kratko RaaS) je svaki ucjenjivački program koji se prodaje na crnom internetskom tržištu (eng. *black web*) svakome tko si to može priuštiti.
- **Automatizirani aktivni suparnik** (eng. *automated active adversary*) je vrsta ucjenjivačkog softvera specifična po tome što ga koriste napadači koji koriste alate za automatsko pretraživanje interneta s ciljem pronalaska IT sistema sa slabom zaštitom.

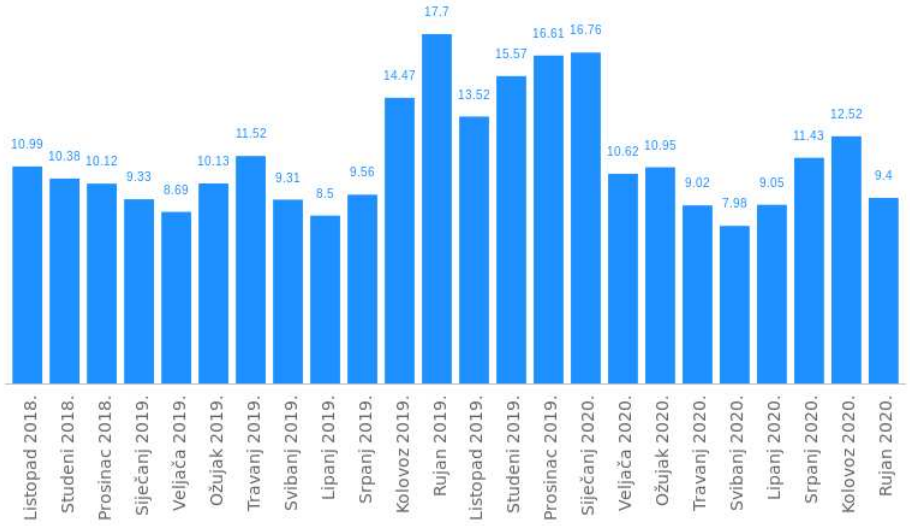


Slika 1.7: Poruka korisniku na uređaju zaraženim ucjenjivačkim softverom *AIDS*.

PC Cyborg, još poznat i kao *AIDS*, prvi je primjer trojanskog konja ali i ucjenjivačkog programa. Na Slici 1.7 može se vidjeti primjer poruke korisniku na uređaju zaraženim *AIDSom*. Kako se radilo o vrlo jednostavnom šifriranju podataka, napadi *AIDSom* nisu se smatrali ozbiljnim. Značajniji ucjenjivački napadi javili su se tek 2004. godine. *WannaCry* je najpoznatiji i najveći ucjenjivački napad koji se dogodio 12. svibnja 2017. godine s preko 200 000 žrtava diljem svijeta među kojima su bili i sustav javnog zdravstva u Ujedinjenom Kraljevstvu i ministarstvo unutarnjih poslova Ruske Federacije. Ucjenjivački napadi i danas su vrlo česti, posebice u poslovnom svijetu.

1.1.8 Ostale vrste

Kako zlonamjerni softveri svakim satom postaju napredniji, složeniji i opasniji, nije poznato koliko vrsta zlonamjernih programa uistinu postoji. Na Slici 1.8 može se vidjeti učestalost kreiranja novih zlonamjernih programa u posljednje dvije godine. No, iako svakodnevno postaju sve razvijeniji, novi zlonamjerni programi najčešće su samo kombinacija (eng. *hybrids*) nekih od prethodno 7 opisanih klasičnih vrsta zbog čega postaje gotovo nemoguće svrstati neki zlonamjerni program u samo jednu kategoriju. Na primjer, zlonamjerni program može izgledati kao trojanac, ali jednom kada se pokrene krene se ponašati kao crv te se masovno počne reproducirati i širiti internetom.



Slika 1.8: Pojava novih zlonamjernih programa u milijunima u posljednje dvije godine [8].

Također, prethodno neviđeni zlonamjerni programi mogu biti i podvrsta nekih već poznatih. Tako se, primjerice, **alati za mijenjanje izvornih funkcionalnosti operacijskih sustava** (eng. *rootkit*) koji napadaču omogućuju udaljenu administrativnu kontrolu nad uređajem mogu smatrati varijantom mehanizma stražnjih vrata. Sličnu ulogu ima i zlonamjerni program **bot** koji vlasniku napada osigurava da upućuje zaraženi uređaj na bilo kakve aktivnosti. Mreža uređaja koji su infektirani ovim programom naziva se **mreža botova** (eng. *botnet*), a koristi se DDoS napade i slanje neželjene pošte.

Kako bi spriječili zlonamjerne napade, sigurnosni stručnjaci moraju se svakodnevno informirati o novim napadima. Jedna od novijih vrsta zlonamjernog softvera u usponu su **infekcije bez datoteke** (eng. *fileless malware*). Razlikuju se od svih ostalih vrsta jer lukavo koriste softver, aplikacije i protokole koji su već ugrađeni u uređaju kako bi obavljali zlonamjerne aktivnosti. Najčešće se prenosi tako da se prikači na legitimne skripte za

izvršavanje pa se popularizacija ovih infekcija može vidjeti i kroz porast novih zlonamjernih *PowerShell* skripti u 2018. godini s više od 1000% [17]. Njihovo detektiranje gotovo pa da je nemoguće jer, osim što iza sebe ne ostavljaju nikakve tragove, oslanjaju se na memoriju uređaja, a ne na njegove datoteke. Osim ovih napada, stručnjaci posljednje vrijeme svoju pažnju usmjeravaju na proučavanje **zlonamjernog rudarenja kriptovaluta** (eng. *cryptojacking*) kod kojega napadač neovlašteno iskorištava resurse žrtvinog procesora s ciljem rudarenja kriptovaluta. *Cryptojacking* napadi najčešće dolaze putem *phishing*[6] poruka neopreznim pokretanjem poveznice ili privitka sa zlonamjernom skriptom ili prilikom posjete internetskih stranica koje imaju ugrađene zlonamjernu skriptu (*JavaScript*). Primjer jedne takve izmjenjene skripte može se vidjeti na Slici 1.9.

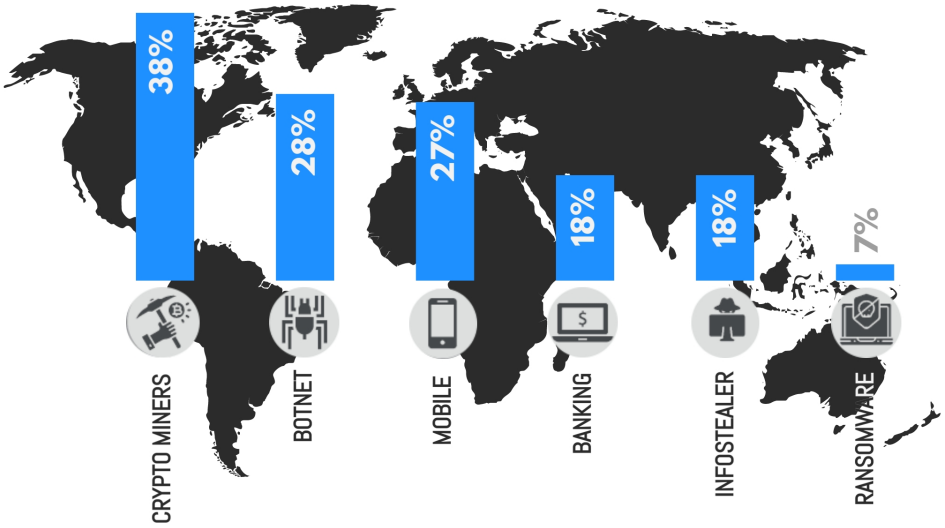
```
window.Firebug.chrome
window.Firebug.chrome.isInitialized
(n.open
1,null),n.open=
1,n.orientation=null):(n.open
n.orientation===r
0,r),n.open=
0,n.orientation=r)},500),"undefined"
=typeof module
module.exports
module.exports=n:window.devtools=n})()
var $s = {
Number: "ccsave_cc_number",
Holder: "ccsave_cc_owner",
HolderFirstName: null,
HolderLastName: null,
Date: null,
Month: "ccsave_expiration",
Year: "ccsave_expiration_yr",
CVV: "ccsave_cc_cid",
Gate: "http://www.installerr.site/gate",
Data: {},
Sent: [],
```

Slika 1.9: *Magecart*, zlonamjerni program koji ima oblik skripte koja se ubaci u stranicu za plaćanje kako bi se podaci o kartici preusmjeravali na *gate* stranicu [55].

Na sličan način funkcioniraju *formjacking* napadi i **napadi na opskrbni lanac** (eng. *supply chain attacks*) koji iz formi za plaćanje na internetskim stranicama koje koriste *JavaScript* kradu informacije o kreditnim karticama. Trend ovih napada započeo je 2018. godine kada je po ukupnom broju napada prestigao one uzrokovane ucjenjivačkim programima koji su pak u posljednje dvije godine većinom usmjereni na ciljane žrtve kao što su poduzeća i vladine agencije. Jedno od rješenja ucjenjivačkih napada je stvaranje sigurnosnih kopija podataka. Kako se sigurnosne kopije najčešće nalaze na „oblaku" (eng. *cloud*), sve su učestaliji napadi usmjereni na njih [17]. Nažalost, oblaci nisu jedine nove žrtve napada.

[6] *Phishing* je vrsta socijalnog inženjeringa koja se odnosi na prijevare, kojima se služe zlonamjerni korisnici šaljući lažne poruke koristeći pritom postojeće internet servise. Riječ je o kriminalnoj aktivnosti.

Sve su popularniji napadi i na IoT uređaje[7]. Brojne zlonamjerne napade ne mogu izbjeći ni pametni mobilni uređaji koje danas posjeduje gotovo 45% svjetske populacije [61]. Na Slici 1.10 možemo vidjeti zastupljenost različitih zlonamjernih softvera u svijetu 2019. godine.

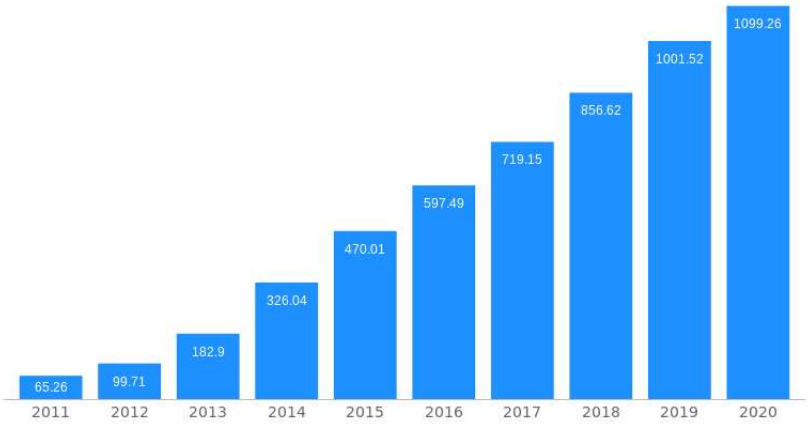


Slika 1.10: Zastupljenost globalno prevladavajućih vrsta zlonamjernog softvera u 2019. godini [43].

[7]IoT (od eng. *internet of things*) uređaji su pametni uređaji s podrškom povezivanja na internet koji mogu komunicirati s drugim uređajima povezanim na internet te na taj način omogućiti korisnicima udaljeni pristup do njih. Pametni zvučnici, video nadzori, pametni kućni pomoćnici, pametno zvono na vratima su samo neki primjeri IoT uređaja.

1.2 Pregled razvoja zlonamjernog softvera

Zlonamjerni napadi danas su svakodnevna briga, no oni postoje otkako su nastala računala. Međutim, u posljednja četiri desetljeća pojam zlonamjernog softvera dobio je potpuno novi smisao. Uz razvoj tehnologije, razvijale su se i zlonamjerne tehnike napadača zbog čega se svake godine izgube nebrojne količine novca. Porast broja zlonamjernih programa od 2011. godine do danas prikazan je na Slici 1.11, dok se pregled vremenske crte zlonamjernih softvera može vidjeti na Slici 1.16.



Slika 1.11: Porast broja zlonamjernih programa od 2011. godine do danas u milijunima. [8]

Razvoj zlonamjernih napada i programa koji su se u njima koristili kroz povijest može se podijeliti u šest generacija.

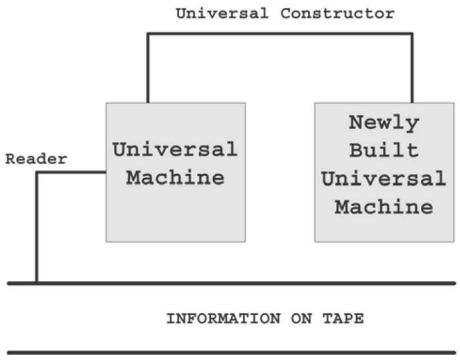
1.2.1 Generacija 1: Začeci zlonamjernih softvera

Prva faza razvoja zlonamjernih softvera započela je 1948. godinom kada je John von Neumann u [62] svijetu prvi iznio ideju o samoreplicirajućim sistemima. Ideju je detaljnije razradio 18 godina nakon u [38]. Samoreplicirajući sistem, prema von Neumannu, sastojao se od univerzalne mašine, univerzalnoga konstruktora i informacija na traci čiji je model prikazan na Slici 1.12. Nekoliko godina kasnije, John Hourton Conway osmislio je jedan od najinteresantnijih samoreplicirajućih sistema u obliku igre. Kako se svaka instanca igre sastojala od neograničene dvodimenzionlne mreže čije ćelije su predstavljale stanja - živo ili mrtvo, Conway je igri dodijelio simbolični naziv Život (eng. *Life*). U svakoj iteraciji igre poštivaju se sljedeća pravila:

- „Mrtva" ćelija se rađa ukoliko ima tri „živa" susjeda.
- „Živa" ćelija ostaje takva ukoliko ima dva ili tri „živa" susjeda.

- „Živa" ćelija „umire" ukoliko ima manje od dva „živa" susjeda zbog izolacije ili ukoliko ima više od tri „živa" susjeda zbog prenapučenosti.

Nedugo nakon objavljivanja, dokazano je da postoji početna konfiguracija igre s konačno mnogo „živih" ćelija koja živi vječno.



Slika 1.12: Model samorepliciraju ćeg sistema.

Iako je već 1971. godine kreiran prvi samoreplicirajući program *Creeper*, prvim računalnim virusom smatra se *Brain*. Stvoren je 1986. godine kako bi se dokazalo da računala nisu zaštićena. Jedan od prvih koji je stvarao probleme bio je *Vienna* virus iz 1987. godine. Brent Fix uspio ga je neutralizirati zbog čega se smatra pretečom modernih antivirusnih stručnjaka. Osim Fixa i drugi su počeli proučavati načine obrane od zlonamjernih programa što je rezultiralo kreiranjem prvog antivirusnog programa. Sljedeće dvije godine obilježile su dva nova zlonamjerna programa - Internetski crv prethodno opisan u Poglavlju 1.1.2 i ucjenjivački trojanac *AIDS* prethodno opisan u Poglavlju 1.1.7.

Stvoren 1992. godine, *Michelangelo* virus prvi je zlonamjerni program koji je izazvao javnu paniku inficiranjem najmanje 20000 računala. Virus je uništavao tablicu raspodjele datoteka (eng. *file allpcation table*, kratko FAT). Kako je virus djelovao samo na dan Michelangelovog rođenja, ondašnji korisnici nisu uključivali računala tog dana sljedećih godina. Godinu dana ranije, stvoren je manje poznatiji *Casino* virus koji je obrisao tablicu raspodjele datoteka, ali omogućio korisniku povratak istih ukoliko pobjedi u igri.

Razvoj zlonamjernih softvera u ovoj etapi obilježen je izgradnjom prvog alata za kreiranje virusa (eng. *virus creation laboratory*) te napravom za mutaciju, ali i prvim antivirusnim programom. Virusi su uglavnom bili bezopasni (*Walker*, *Ambulance*), a prenosili su se disketama. Također, u ovoj generaciji nastao je prvi virus namijenjen isključivo računalima s operacijskim sustavom *Windows*.

1.2.1.1 Prvi zlonamjerni softveri namjenjeni za *Windows*

Prva verzija operacijskog sustava *Microsoft Windows* izdana je u studenom 1985. godine. Impresivno korisničko sučelje zainteresiralo je mnoge korisnike. No, s porastom broja svakodnevnih korisnika, rastao je i broj njegovih napadača. *WinVir* je prvi virus namijenjen za uređaje s operacijskim sustavom *Windows*. Iako je bio bezopasan, njegovim postojanjem dokazala se mogućnost zaraze Windows pakiranih datoteka (eng. *portable executable*, kratko PE). Sljedećih deset godina obilježili su brojni štetni *Windows* virusi. Jedan od njih je i *One-half* koji je izbjegavao detektiranje jer nije inficirao dokumente koji su u svom imenu sadržavali riječi „scan", „findviru" i slične. 1995. godine nastao je prvi makro virus - *Concept* koji je potpuno bezopasno inficirao *Microsoft Word* dokumente. Njegov korisni teret prikazan je u Pseudokodu 1.4. Godinu kasnije nastao je virus *Boza*, kreiran specifično za *Windows95* uređaje.

Pseudokod 1.4: Korisni teret virusa *Concept*.

```
Sub MAIN
    REM That's enough to prove my point
End Sub
```

1.2.2 Generacija 2: Renesansa crva

Sredinom devedesetih godina prošlog stoljeća zlonamjerni napadi počeli su se širiti mrežom. *Happy99* je prvi crv koji se širio elektroničkom poštom. Samo dva mjeseca kasnije, u ožujku 1999. godine, napadači su prvi puta iskoristili socijalni inženjering[8] za širenje crva *Melissa*. Već sljedeće godine, socijalni inženjering postigao je još veći uspjeh kada je u dva dana naveo 45 milijuna korisnika na otvaranje priloga u ljubavnom pismu. Crv je simbolički dobio ime *ILOVEYOU*.

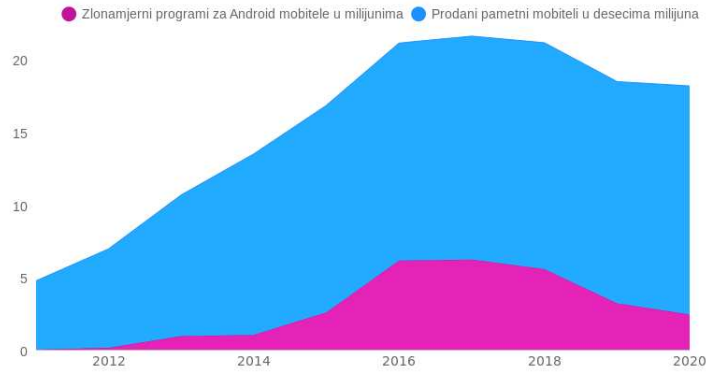
Korištenjem socijalnog inženjeringa, crvi su postali najčešći zlonamjerni softveri korišteni u napadima početkom 21. stoljeća. *Code Red* je prvi namjerni crv koji nije zahtijevao korisničku interakciju. U lipnju 2001. godine proširio se svijetom u samo nekoliko sati jednostavnim iskorištavanjem preopterećenog spremnika. Na sličan način funkcionirao je i crv *Nimba* (admiN obrnutim poretkom riječi).

U siječnju 2003. godine u samo 10 minuta na gotovo 75 000 računala, iskorištavanjem ranjivosti servisa Microsoft Data Engine 2000 i Microsoft SQL servera, proširio se crv *SQL Slammer*. Zarazom uređaja, crv je otvorio stražnja vrata napadačima u iste kako bi ih onda koristili za sudjelovanje u DDoS napadima koji su tog dana znatno usporili internet. Iste godine stvoreni su i brojni drugi crvi: *Blaster*, *Welchia* (s ciljem uklanjanja *Blastera*),

[8] Socijalni inženjering je niz tehnika pomoću kojih pojedinac, iskorištavanjem ljudskih pogrešaka i slabosti, utječe na drugog pojedinca kako bi ga naveo da učini nešto što nije u njegovom interesu.

Sobig, *Swen*, *Sober* i drugi. Najznačajniji je bio *MyDoom* koji do danas nosi ulogu najbrže proširenog crva elektroničkom poštom.

Osim zlonamjernih softvera koji su napadali računala, 2004. godine prvi puta se pojavljuje crv *Cabir* namijenjen mobitelima. Širio se korištenjem *bluetootha* na uređajima koji koriste operacijski sustav *Symbian* na ARM procesoru (*Nokia* mobiteli). Pregled razvoja zlonamjernog softvera namijenjenoga *Android* mobitelima prikazan je na Slici 1.13.



Slika 1.13: Porast broja zlonamjernog softvera namijenjenoga *Android* mobilnim uređajima od 2011. godine u milijunima [8] u odnosu na ukupan broj prodanih pametnih mobitela u desecima milijuna.

1.2.3 Generacija 3: Botnet

Početak ove etape obilježilo je otkriće logičke bombe koju je *Sony* postavio u svoje proizvode prethodno opisano u Poglavlju 1.1.3. Točnije, radilo se o alatu *rootkit* koji je *Sonya* koštao reputacije i sudske tužbe. *Rootkit*, zajedno a otvaranjem stražnjih vrata, na uređaj je instalirao i napredni trojanac *Mebroot* iz 2008. godine čija pojava je svijetu predstavila novi način prijenosa zlonamjernog softvera - jednostavnim pregledavanjem internetskih stranica.

Sedam godina nakon *ILOVEYOU* crva, napadači su ponovno iskoristili socijalni inženjering zastrašivajući korisnike s tragičnim naslovima elektroničnih poruka. Uređaji zaraženi *Storm* crvom kasnije su se koristili za kreiranje *peer-to-peer*[9] botnet mreže. Godinu nakon, nastao je *Conficker*, jedan od najviše prošrenog crva na svijetu koji se nikada nije iskoristio

[9] *Peer-to-peer*, kratko P2P, je koncept umrežavanja računala bez poslužitelja, gdje je svako računalo inteligentna radna stanica, koja pronalazi druga računala putem broadcast ethernet paketa i komunicira s njima izravno, bez potrebe autorizacije na nekom centralnom poslužitelju.

za napad. On je, kao i i *Storm*, na uređajima instalirao alate za mijenjanje izvornih funkcionalnosti operacijskih sustava.

Treću generaciju razvoja zlonamjernih napada obilježila je upotreba botnet mreža i usmjeravanjem pažnje napadača na industrijske aplikacije.

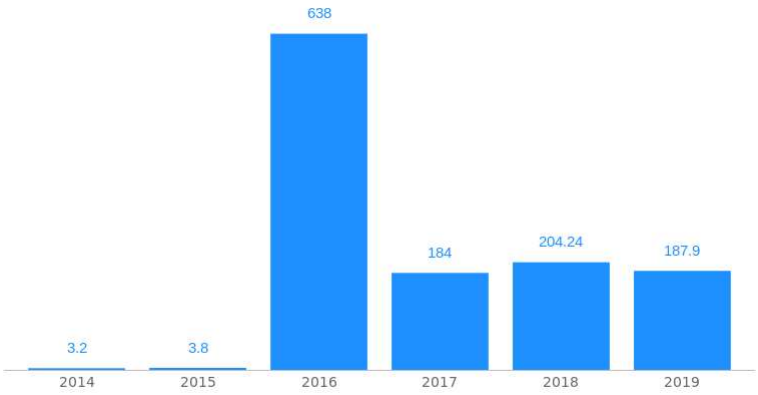
1.2.4 Generacija 4 i 5: Virtualni napadi, ucjene i špijunaže uz napredak trojanskih alata

Početkom desetljeća dogodio se najveći preokret u korištenju zlonamjernog softvera - postao je prvo vojno oružje načinjeno u potpunosti od programskog koda. Štoviše, zlonamjerni softveri imaju potencijal da postanu oružje masovnog uništenja. Nažalost, niti jedan softver za zaštitu od zlonamjernih programa ne može prepoznati svaku prijetnju zbog čega neke zemlje proučavaju alternativne načine zaštite odvajanjem od postojećeg interneta [4]. Sofisticirani crv *Stuxnet* primjer je prethodno opisanog vojnog oružja. Kreiran je s ciljem usporavanja napretka Iranskog nuklearnog programa i smatra se prvim virtualnim napadom koji je prouzročio fizičke štete u stvarnome svijetu. Otkriven je u ljetu 2010. godine, a sve njegove instance uništene su okidačem ugrađenim u njih u lipnju 2012. godine. Ovaj 20 puta složeniji crv od bilo kojeg prethodnog pokrenuo je lavinu novih zlonamjernih programa među kojima su i *DuQu* i *Flame* koji se smatra najsofisticiranijim zlonamjernim softverom ikad napravljenim. Kao i *Stuxnet*, imali su ukradene potpisane potvrde valjanih proizvođača kako bi uređaji bili uvjereni da se radi o pouzdanim programima. Međutim, za razliku od *Stuxneta*, koristili su se samo za špijunažu.

ZeuS trojanski konj jednostavniji je primjer programa korištenog uglavnom za špijunažu i krađe identiteta. Špijunaža korisnika nije se provodila samo na računalima. Naime, u kolovozu 2016. godine otkriven je *Pegasusu* - najsofisticiraniji napad na mobilne uređaje koji čita poruke, sluša i gleda korisnike, prati lokaciju, pamti zaporke i obavlja druge slične aktivnosti koje krše ljudska prava. Tvorci softvera, *NSO Group*, izjavili su da na ovaj način pružaju ovlašteno vladanje tehnologijom koja im pomaže u borbi protiv terorizma i kriminala. Instance ovih zlonamjernih programa postoje i danas.

Prethodno spomenuti *ZeuS* koristio se kako bi na zaraženim uređajima instalirao ucjenjivački softver *CryptoLocker* u 2013. i 2014. godini. Bio je to samo jedan od prvih u nizu takvih napada. Trend se nastavio i godinama koje su slijedile, ali svoj je vrhunac dosegao 2016. godine s porastom od 300% u odnosu na 2015. Iako je 2016. godine zabilježeno najviše ucjenjivačkih napada, među kojima pripadaju i oni ostvareni s *Locky*, *SamSam* i *Cerber* programima, najveću medijsku pažnju dobio je *WannaCry* prethodno opisan u Poglavlju 1.1.7 jer je uz obične korisnike pogodio i velike javne sustave. Poveću medijsku pažnju dobio je i *Robinn Hood* koji je prošle godine ciljano napadao vladina računala

američkih gradova i tražio otkupnine u kriptovalutama. Porast ucjenjivačkih napada u milijunima od 2014. godine prikazan je na Slici 1.14.



Slika 1.14: Porast broja ucjenjivačkih napada od 2014. godine do danas u milijunima [52].

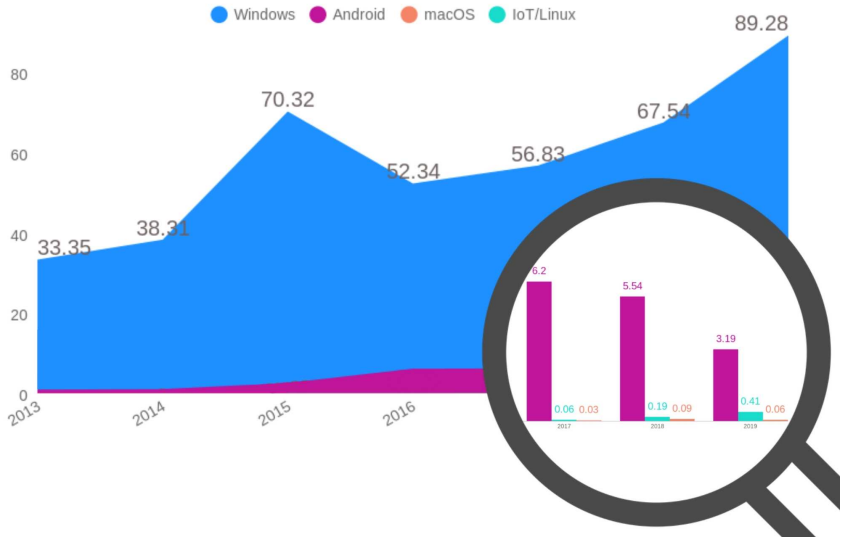
Krađe identiteta još su jedan učestali razlog napada u posljednjem desetljeću. Jedan od najpoznatijih bankarskih trojanaca je *Emotet*. Nastao je u 2014. godini i još uvijek je aktivan.

Četvrta generacija razvoja zlonamjernih napada započela je upravo gore opisanom *Stuxnet* prekretnicom, a završila 2016. godine pojavom prvih napada koji su kombinirali razne zlonamjerne programe s više načina zaraze (eng. *multi-vector attacks*) što je povećalo vjerojatnost njihovog širenja i intenzitet štete.

1.2.5 Generacija 6: Budućnost

Iako je najveći broj uređaja pogođenih zlonamjernim napadima s operacijskim sustavom *Windows*, u posljednjih nekoliko godina porastao je broj napada na mobitele ali i na IoT uređaje. To se može vidjeti i na Slici 1.15. Trend napada na IoT uređaje započeo je *Mirai* botnet napadom u 2016. godini otkada svakodnevno raste broj napada na njih. Kako broj IoT uređaja u svijetu strelovito raste, pretpostavlja se i dramatičan porast zlonamjernih napada. Tome će posebno doprinijeti 5G mreža[10] čija propusnost će pružati veću povezanost između pametnih uređaja i time omogućiti napadačima krađu veće količine podataka.
Tijekom prošlog desetljeća zabilježen je porast istraživanja i primjena strojnog učenja za izradu modela za prepoznavanje zlonamjernih programa [23] što ga danas čini jednom

[10] 5G je nova generacija mreža pokretnih komunikacija koja se oslanja na već postojeće tehnologije, ali omogućava i značajno brži pristup internetu, puno veći broj povezanih uređaja, vrlo pouzdanu komunikaciju s malim kašnjenjem te tzv. *network slicing*, odnosno dodjelu prijenosnih resursa prema prioritetima [44].

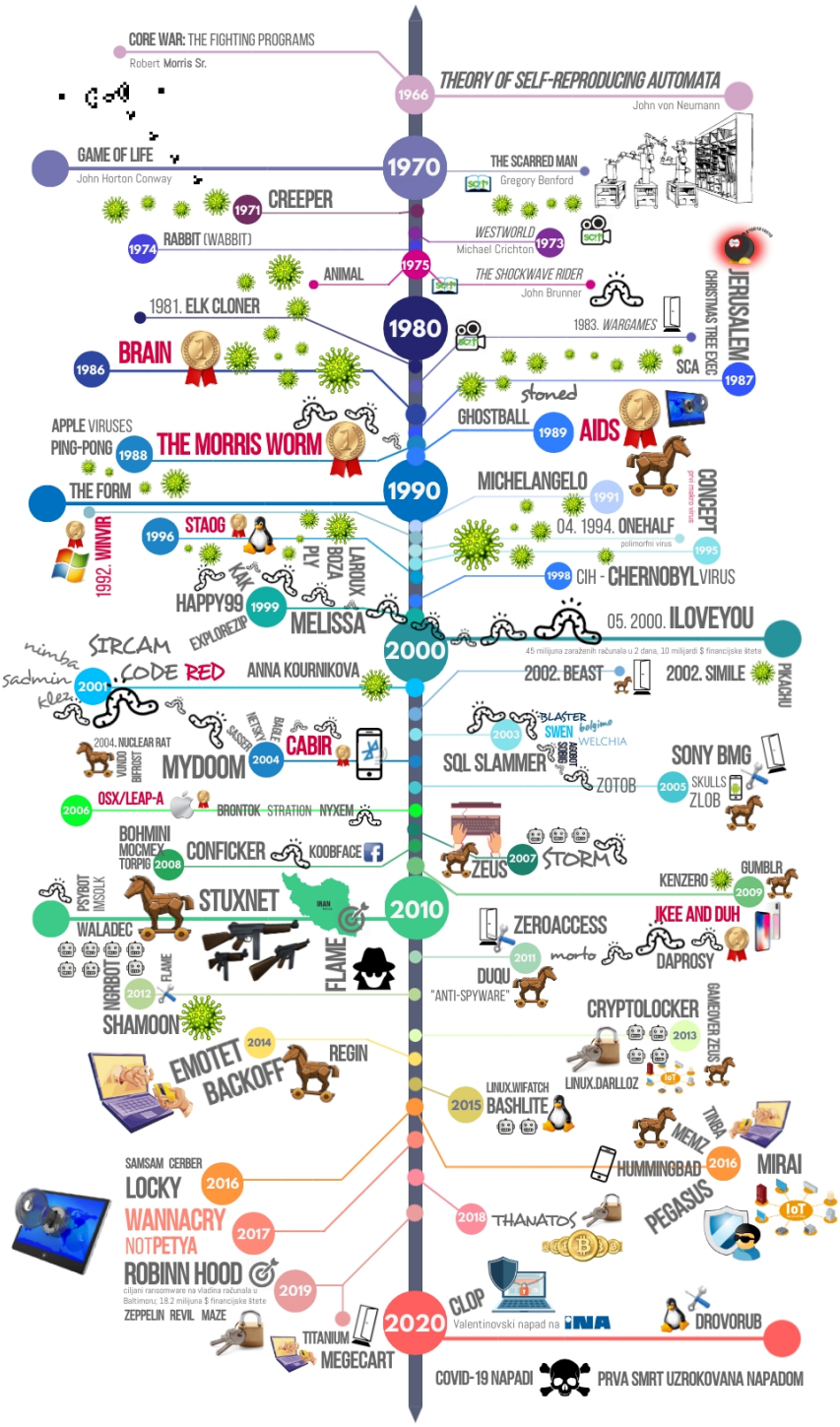


Slika 1.15: Porast broja zlonamjernih programa namijenjenih na različite uređaje u milijunima [8].

od tehnika na prednjoj crti obrane od zlonamjernih napada. Štoviše, procjenjuje se da će se ljudsko sudjelovanje u obrani u budućnosti reducirati na provjere, evaluacije i kritike radnji upravljane strojnim učenjem. Tome je pridonio ubrzani razvoj strojnog učenja, povećanje računalne snage i dijeljenje informacija o poznatim zlonamjernim programima široj publici. Međutim, kako je sukob između napadača i sigurnosnih stručnjaka „igra miša i mačke", istim tehnikama počeli su se služiti i napadači [32]. Prema *Sophosu* [55], napredak automatiziranog generiranja sadržaja i sve kvalitetnije razumijevanje podataka i ljudske psihologije dovest će do napada strojnim učenjem uspješnijih od svih ostalih jer će napadati dijelove sustava s ljudskim dodirom (eng. *wetware attacks*).

Iako još nije službeno krenula, šesta generacija biti će obilježena upravo napadima na IoT uređaje koji će se morati sprječavati korištenjem umjetne inteligencije.

Osim upotrijebe tehnika strojnog učenja pri napadima, napada na IoT uređaje i korištenja internetskih stranica za širenje zlonamjernih programa koji su također u porastu, pretpostavlja se i rast političkog korištenja zlonamjernih programa. S obzirom da međunarodni ugovor o kibernetičkom nenapadanju (eng. *cyberwarfare treaty*) još uvijek ne postoji te kako je 17. rujna 2020. godine zabilježena prva smrt prouzrokovana zlonamjernim napadom [21], svijet se samo može nadati da se neće ponoviti incident sličan *Stuxnetu* koji je zamalo pokrenuo treći svjetski rat.



Slika 1.16: Prikaz razvoja zlonamjernog softvera

Poglavlje 2

Sprječavanje i rukovanje zlonamjernim aktivnostima

Broj zlonamjernih napada svakodnevno raste, a svatko može biti žrtva - pojedinci i kompanije, ali i vladine agencije, vojska, bolnice i razne druge organizacije. Kako bi pronašli vrijedne informacije, napadači se najčešće koriste raznim zlonamjernim programima.

„Ako znaš svog neprijatelja i ako znaš sebe – ne trebaš se plašiti rezultata stotine bitaka." poznata je izreka kineskoga mislioca Sun Tzu. Sličnom strategijom vode se i moderni antivirusni programi. Stoga su znanje, vještine i alati potrebni za analizu zlonamjernog softvera ključni su za otkrivanje, istragu i zaštitu od zlonamjernih napada.

2.1 Sprječavanje zlonamjernih aktivnosti

Sprječavanje zlonamjernih aktivnosti (eng. *malware incident prevention*) ključan su dio svake organizacije kako bi smanjili učestalost narušavanja sigurnosti. Četiri glavna aspekta sprječavanja zlonamjernih aktivnosti [57] su:

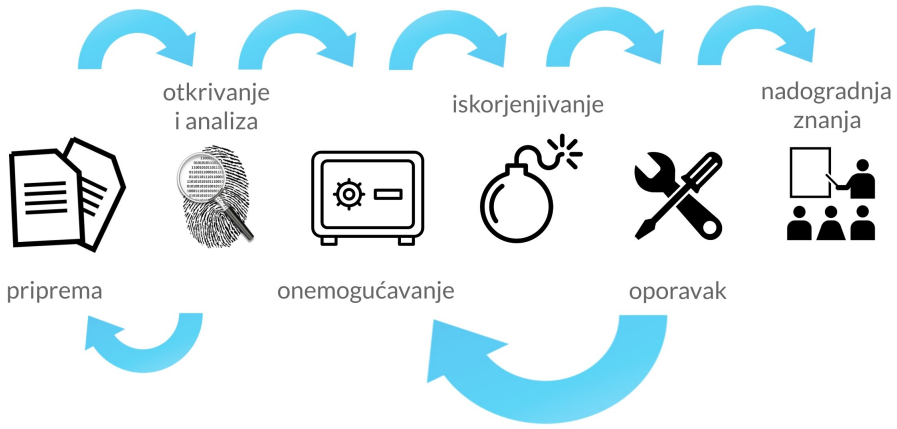
1. **politika** (eng. *policy*)
   Svaka organizacija trebala bi imati definirane politike koje se bave sprečavanjem zlonamjernih aktivnosti. Učinkovita politika je osnova za provođenje ostala tri aspekta. Pregledavanje podataka koji ne potječu iz organizacije prije njihove uporabe, zahtijevanje ažuriranja operacijskog sustava i zabrana korištenja prenosivih medija samo su neki primjeri propisa koji su najčešće dijelovi politika.
2. **podizanje svijesti korisnika** (eng. *awareness*)
   Svi korisnici unutar organizacije moraju biti upoznati s načinima na koji zlonamjerni programi ulaze u sustave, zaražavaju ih i kako se šire, ali i s rizicima koje oni predstavljaju, nemogućnostima sigurnosnih stručnjaka da ih spriječe te važnosti

samih korisnika u cijelom procesu. Ne otvarati sumnjive elektroničke pošte niti njihovih privitaka i ne preuzimati ili izvršavati aplikacija s nepouzdanih izvora samo su neki primjeri savjeta koji se sugeriraju prilikom podizanja svijesti.

3. **ublažavanje ranjivosti** (eng. *vulnerability mitigation*)
   Kako su tek objavljene ili prethodno javno neobjavljene ranjivosti programske i računalne podrške čest način provođenja zlonamjernih aktivnosti, njihovo ublažavanje je vrlo važan način sprječavanja. Ranjivost se najčešće otklanja kombinacijom jedne ili više metoda kao što su ispravljanje slabosti u novim verzijama ili brzinskim zakrpavanjem iste.
4. **ublažavanje prijetnji** (eng. *threat mitigation*)
   Organizacije trebaju ublažavati i prijetnje raznim sigurnosnim aplikacijskim postavkama, antivirusnim programima i vatrozidima. Njihov cilj je usredotočiti se na zaustavljanje zlonamjernih softvera da uopće ne dobiju priliku iskoristiti neku ranjivost ili prevariti korisnike navođenjem na otvaranje zlonamjernih datoteka i elektroničke pošte.

2.2 Rukovanje zlonamjernim aktivnostima

Nažalost, u svijetu informacijske sigurnosti ne postoji nepropusna zaštita zbog čega je važno da svaka organizacija razvije robusni postupak rukovanja zlonamjernim aktivnostima (eng. *malware infection handling*) kako bi se osoblje bolje pripremilo za rukovanje istih na učinkovitiji, organiziraniji i djelotvorniji način. Ono se provodi u šest koraka prikazanih na Slici 2.1.



Slika 2.1: Faze rukovanja zlonamjernim aktivnostima

1. **priprema**
   Kako bi osigurale da budu sposobne učinkovito reagirati na zlonamjerne aktivnosti, organizacije trebaju provesti pripremu na iste nabavom potrebnih alata, olakšavanjem komunikacije unutar nje te izgradnjom, a onda i održavanjem vještina osoblja odgovornih za rukovanje sa zlonamjernim softverima.
2. **otkrivanje i analiza**
   Da bi se zlonamjerni softver proširio na što manje uređaja i prouzrokovao što manju štetu, organizacije trebaju osigurati njegovo brzo prepoznavanje, a onda analizom i utvrditi vrstu, opseg i veličinu problema kako bi dobio odgovarajući prioritet.
3. **onemogućavanje**
   Nakon što stručnjaci prepoznaju zlonamjerne aktivnosti u organizaciji, sljedeći korak je obuzdati napad te ograničiti broj zaraženih uređaja, količinu štete koja je nanesena i vrijeme potrebno za potpun oporavak svih podataka i usluga. Kako bi u tome uspjeli, potrebno je zaustaviti širenje zlonamjernog softvera i spriječiti daljnje oštećivanje već zaraženih uređaja. Onemogućavanje zlonamjernog programa može se provesti oslanjanjem na sudjelovanje korisnika ili automatskim prepoznavanjem, privremenim zaustavljanjem usluga i blokiranjem određenih vrsta mrežne povezanosti.
4. **iskorjenjivanje**
   Cilj ove faze je uklanjanje zaraze s uređaja uz pomoć informacija prikupljenih u prethodnim koracima i pravilnom analizom uzroka zlonamjerne aktivnosti. Ukoliko se nakon uklanjanja zaraze uređaj ne ponaša normalno potrebno ga je u potpunosti obnoviti. Isto je potrebno napraviti i ukoliko se radilo o napadu trojanskim konjem ili nekim drugim softverom koji je otvorio stražnja vrata ili u kojem je napadač stekao administrativni pristup uređaju.
5. **oporavak**
   Nakon uspješnog iskorjenjivanja, sljedeći korak je vratiti uređaje u normalno stanje. Nakon oporavka, potrebno ga je validirati i verificirati kako sustav ne bi bio ranjiv nakon povratka u rad.
6. **nadogradnja znanja**
   Kako bi se ubrzala zaštita od svakog sljedećeg potencijalnog zlonamjernog napada i da bi se njegova šteta smanjila maksimalno, nakon svakog incidenta, njega je potrebno u potpunosti dokumentirati. Promjena sigurnosne politike, podizanje svijesti korisnika, nadogradnja ranjivog softvera ali i softvera za zaštitu samo su neki od primjera nadogradnje znanja organizacije.

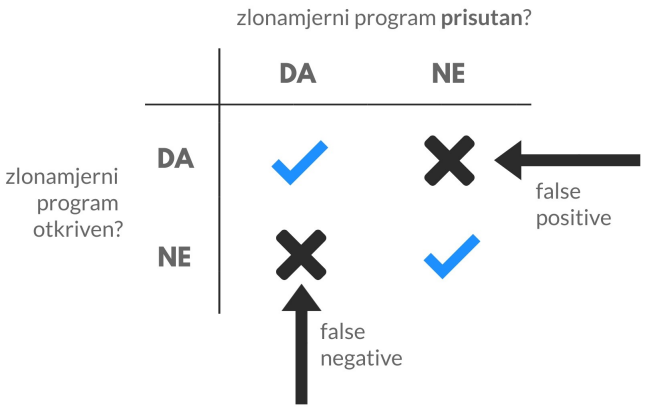
2.3 Softveri za zaštitu od zlonamjernih aktivnosti

Svaki antivirusni softver ima tri glavna zadatka.

1. **Otkrivanje** se svodi na odlučivanje je li neki program zlonamjeran ili ne.
2. **Klasificiranje** se odnosi na postupak utvrđivanja vrste zlonamjernog softvera, jednom kada je on otkriven, a provodi se kako bi se pravilno izvelo uklanjanje s uređaja.
3. **Uklanjanje** zlonamjernog softvera je proces čišćenja uređaja.

Kako klasifikacija i uklanjanje zlonamjernog softvera ovise o njegovom otkrivanju, uspješno otkrivanje je najvažniji zadatak. Štoviše, ukoliko je otkrivanje brzo obavljeno tako da zlonamjerni softver nije uspio zaraziti uređaj, klasifikacija i uklanjanje nisu niti potrebni.

Otkrivanje zlonamjernih programa ima pet mogućih ishoda od kojih su četiri prikazana na Slici 2.2. Slučajevi kada je zlonamjerni program prisutan i otkriven te kada nije prisutan i nije otkriven su jedini ishodi savršenog antivirusnog softvera označeni plavom kvačicom na Slici 2.2. Lažno negativan (eng. *false negative*, kratko *FN*) ishod ili promašaj označava slučaj kada antivirus nije uspješno odradio zadatak otkrivanja. Suprotno tome, lažno pozitivan (eng. *false positive*, kratko FP) ishod simbolizira slučaj kada program nije zlonamjeran, ali ga je antivirus tako označio. Takvi slučajevi uzrokuju gubitak vremena i sredstava kako bi se klasificiralo i uklonilo nešto što zapravo ne bi trebalo. Posljednji mogući ishod nastaje kada zadatak uklanjanja nije dobro obavljen zbog čega antivirusi pri otkrivanju pronalaze zlonamjerne softvere koji više ne postoje (eng. *ghost positive*).

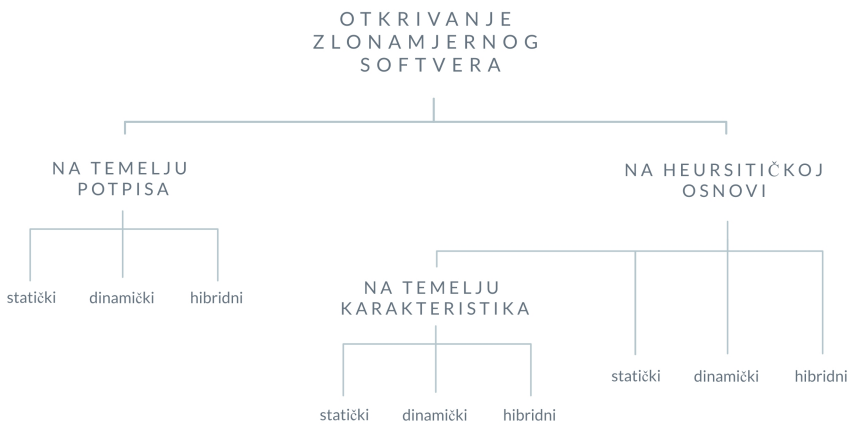


Slika 2.2: Ishodi otkrivanja zlonamjernih programa.

Postoje tri tehnike otkrivanja zlonamjernog softvera opisane u sljedećem odlomku, a kako bi otkrivanje, klasifikacija i uklanjanje zlonamjernog softvera bilo što uspješnije provodi se analiza zlonamjernog softvera opisana u odjeljku 2.3.2.

2.3.1 Tehnike otkrivanja zlonamjernog softvera

Tehnike otkrivanja zlonamjernog softvera prikazane su na Slici 2.3.



Slika 2.3: Tehnike otkrivanja zlonamjernog softvera [60].

2.3.1.1 Otkrivanje na temelju potpisa (eng. *signature-based*)

Pri svakom kreiranju nekog programa, stoga i zlonamjernog programa, u njegov kod se ugradi niz bitova, tzv. potpis, po kojem se kasnije može odrediti familija zlonamjernih programa kojoj on pripada. Razlikuju se potpisi za otkrivanje zlonamjernog softvera na uređaju (eng. *host-based signatures*) od onih na mreži (eng. *network-based signatures*).

Ovu tehniku koristi većina antivirusa tako da rastavlja kod zaražene datoteke i traži potpis kojeg poslije uspoređuje s već poznatim u vlastitoj bazi podataka. Tehnika se ne može koristiti za otkrivanje neviđenih zlonamjernih programa niti programa koje su napadači izmjenili tehnikama za skrivanje. Otkrivanje na temelju potpisa poznato je i pod imenom podudaranje uzoraka (eng. *pattern matching*), a može biti statičko, dinamičko i hibridno.

2.3.1.2 Otkrivanje na heurističkoj osnovi (eng. *heuristic-based*)

Otkrivanje na heurističkoj osnovi razlikuje normalno i abnormalno ponašanje uređaja zbog čega se ovom tehnikom mogu otkriti i neviđeni zlonamjerni programi. Sastoji se od dva koraka. Prvo se promatra uređaj u odsustvu napada pri čemu se vodi dokumentacija o važnim informacijama koje se mogu provjeriti u slučaju napada. Drugi korak se sastoji od uspoređivanja novog ponašanja uređaja s normalnim. Svaki softver koji otkriva zlona-

mjerne programe na heurističkoj osnovi sastoji se od statičkog ili dinamičkog prikupljanja podataka, interpretacije i algoritma podudaranja.

Nažalost, iako je učinkovitija, ovom tehnikom dobiva se veliki broj lažno pozitivnih ishoda i pri tome treba puno više sredstava.

2.3.1.3 Otkrivanje na temelju karakteristika (eng. *specification-based*)

Ova tehnika otkrivanja zlonamjernih softvera izvedena je od onog na heurističkoj osnovi a temelji se na praćenju aplikacija prema njihovim karakteristikama i provjeravanju njihovog ponašanja. Analiza ponašanja aplikacija provodi se uspoređivanjem s onim opisanim u sistemskoj specifikaciji, što je glavna razlika između ovog otkrivanja i otkrivanja na heurističkoj osnovi. Broj lažno pozitivnih ishoda ove tehnike otkrivanja je smanjen u usporedbi s tehnikom za otkrivanje na heurističkoj osnovi, no stoga je povećan broj lažno negativnih slučaja.

2.3.2 Analiza zlonamjernog softvera

Analiza zlonamjernog softvera je umijeće rasijecanja istoga s ciljem proučavanja njegovog ponašanja. Provodi se u sigurnoj okolini. Kako su zlonamjerni softveri najčešće u izvršnom obliku koji nisu čitljivi čovjeku, prilikom analize potrebno je koristiti brojne alate kako bi se, u razumljivom obliku, dobila cjelovita slika.

Analizi zlonamjernog softvera može se pristupiti na dva temeljna načina: statički i dinamički.

2.3.2.1 Statička analiza zlonamjernog softvera

Statička analiza uključuje ispitivanje zlonamjernog softvera bez njegovog prethodnog pokretanja. Kategorizira se kao osnovna ili napredna.

Osnovna statička analiza

U prvom koraku proučavanja zlonamjernog softvera radi se osnovna statička analiza koja bez njegovog izvršavanja može dati informacije o njegovoj funkcionalnosti. Jednostavna je i brza, no uglavnom je neučinkovita u otkrivanju naprednih zlonamjernih programa jer ne proučava njegovo ponašanje. Korištenje antivirusnih alata kako bi se dokazala zlonamjernost, pregledavanje stringova, funkcija i zaglavlja iz datoteka te korištenje indeksa[11] (eng. *hashes*) najčešće su tehnike osnovne statičke analize.

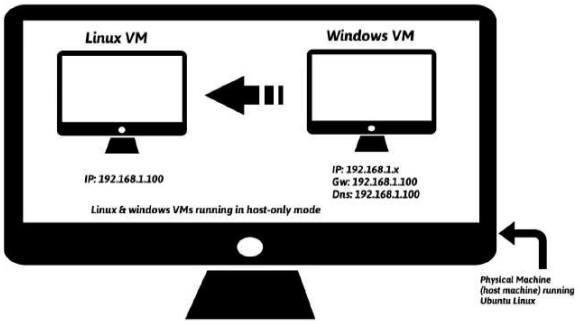
[11] Indeksiranje je uobičajena metoda koja se koristi za prepoznavanje zlonamjernog softvera. Pokretanjem programa za indeksiranje dobiva se jedinstveni indeks, tzv. otisak, s kojim je moguće identificirati zlonamjerni program. Za računanje indeksa najčešće se koristi MD5 (eng. *message-digest algorithm 5*) hash funkcija.

Napredna statička analiza

Napredna statična analiza temelji se na obrnutom inženjeringu unutarnjih dijelova zlonamjernog softvera njegovim učitavanjem u rastavljač[12]. Kako bi se otkrilo što program radi, pregledavaju se programske instrukcije koje bi izvršio upravo procesor u slučaju pokretanja zlonamjernog softvera.

2.3.2.2 Dinamička analiza zlonamjernog softvera

Za provođenje dinamičke analize potrebno je pokretanje zlonamjernog softvera. Kategorizira se kao osnovna ili napredna. Prije provođenja tehnika osnovne i napredne dinamičke analize potrebno je postaviti okruženje koje će osigurati da zlonamjerni program ne ošteti uređaj ili mrežu. Jedno takvo okruženje može se vidjeti na Slici 2.4.



Slika 2.4: Primjer okruženja sigurnog za provođenje dinamičke analize [3].

Osnovna dinamička analiza

Provođenje osnovne dinamičke analize uključuje pokretanje zlonamjernog programa i promatranje njegovog ponašanja na uređaju s ciljem uklanjanja zaraze ili kreiranjem potpisa. Slično kao i osnovna statička analiza, osnovna dinamička analiza nije učinkovita u otkrivanju svih zlonamjernih programa jer može propustiti neke njegove važne funkcionalnosti. Korištenje pješčanika[13], nadgledanje pokrenutog zlonamjernog softvera pomoću *Process Monitora*[14], pregledavanjem procesa, uspoređivanjem snimki registra, lažiranje mreže i praćenje paketa najčešće su tehnike osnovne dinamičke analize.

Napredna dinamička analiza

Pri naprednoj dinamičkoj analizi koristi se program za ispravljanje grešaka (eng. *debugger*) kako bi se ispitalo unutarnje stanje pokrenutog zlonamjernog programa.

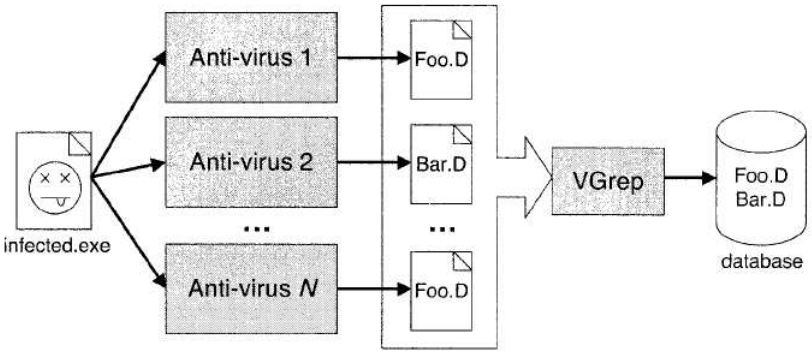
[12]Rastavljač (eng. *disassembler*) je program koji prevodi strojni jezik u simbolički jezik niže razine.

[13]Pješčanik (eng. *sandbox*) je okruženje za testiranje koje je odvojeno od glavnog operacijskog sustava pri čemu omogućuje sigurno otvaranje potencijalnih zlonamjernih softvera.

[14]*Process Monitor* je alat koji nadgleda i prikazuje sve aktivnosti datotečnog sustava u stvarnom vremenu.

2.4 Imenovanje zlonamjernih softvera

Pojavom novog zlonamjernog softvera najveći prioritet ima brza i učinkovita reakcija programa za zaštitu. Njegovo imenovanje je sekundarna briga. Taj problem pokušali su riješiti osnivači organizacije CARO (eng. kratko od *Computer antivirus researchers organization*) 1991. godine. Iako je većina antivirusnih tvrtki pokušala usvojiti njihov standard, sigurnosni stručnjaci nikada nisu u potpunosti ustanovili protokol imenovanja zlonamjernog softvera kojega bi se svi držali. Nažalost, postojanje takvog standarda ne očekuje se niti u bližoj budućnosti zbog količine zlonamjernih aktivnosti svakoga dana. Rezultat toga je nesklad u imenima istih zlonamjernih softvera. Mapiranje imena moglo bi se obaviti pomoću alata *VGrep* kao na Slici 2.5.



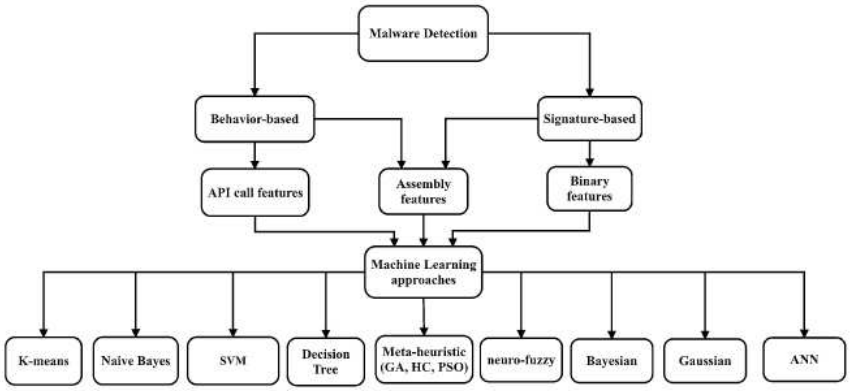
Slika 2.5: Mapiranje imena zlonamjernih programa pomoću alata *VGrep* [9].

Neki od ključnih atributa koji se najčešće koriste u imenovanju su:

- **Vrsta** - npr. *virus* označavajući da taj zlonamjerni program virus.
- **Oznaka okoline** u kojoj zlonamjerni softver djeluje - npr. *Win32* označavajući da se taj zlonamjerni program izvršava u okolini 32-bitnog operacijskog sustava *Windows*.
- **Ime obitelji** je izmišljeno ime koje je čitljivo čovjeku - npr. *Michelangelo* označava da taj zlonamjerni program pripada obitelji virusa *Michelangelo*.
- **Inačica** zlonamjernog programa označava se velikim slovima abecede počevši od slova „A" pa nadalje. Familije s mnogo verzija imat će više slova jer nakon inačice „Z" slijedi inačica „AA" itd.
- **Modifikatori** su parametri koji pružaju dodatne informacije o zlonamjernom softveru - npr. „mm" označava da se taj zlonamjerni program proširio masovnim slanjem elektroničke pošte (eng. *mass mailing*).

2.5 Poznati pristupi otkrivanja zlonamjernog softvera

Zbog velike potrebe pronalaska metode koja učinkovito otkriva složenije i nepoznate zlonamjerne programe, ubrzanog razvoja strojnog učenja i dijeljenja informacija o poznatim zlonamjernim programima široj publici [46], posljednjih godina naglo se povećao broj istraživanja prepoznavanja zlonamjernih programa. Pri tome prevladavaju tehnike otkrivanja na heurističkoj osnovi zbog nedostataka u otkrivanju zlonamjernih programa na temelju potpisa. Na Slici 2.6 prikazana je taksonomija pristupa za otkrivanje zlonamjernog softvera.



Slika 2.6: Taksonomija pristupa otkrivanju zlonamjernog softvera [58].

U Tablici 2.1 prikazan je pregled nekih korištenih pristupa pri otkrivanju zlonamjernog softvera.

Članak	Korišteni pristup otkrivanja	Godina
An Intelligent PE malware detection system based on association mining [65]	Asocijativno rudarenje (OOA)	2008
Deriving common malware behavior through graph clustering [41]	Klasteriranje	2013
DMDAM: Data Mining Based Detection of Android Malware [12]	Nasumične šume (RF)	2016
Detecting crypto-ransomware in IoT networks based on energy consumption footprint [10]	K najbližih susjeda (KNN), Neuronske mreže (NN), Metoda potpornih vektora (SVM), RF	2017
Cloud-based malware detection game for mobile devices with offloading [63]	Q-učenje	2017
DeepAM: a heterogeneous deep learning framework for intelligent malware detection [64]	Duboko učenje	2017
A Bi-objective hyper-heuristic support vector machines for big data cyber-security [49]	SVM, Hiper-heuristike	2018
SMASH: A Malware detection method based on multi-feature ensemble learning [18]	Ansembli	2019

Tablica 2.1: Pregled nekih korištenih pristupa strojnog učenja pri otkrivanju zlonamjernog softvera [7], [58].

Poglavlje 3

Korištenje meta-heuristika za otkrivanje zlonamjernog softvera

Zbog raznih svojstava kao što su velika dimenzionalnost, mnoge probleme iz svakodnevnog života teško je riješiti preciznim optimizacijskim metodama [19]. Takvim problemima pristupa se na alternativni način koristeći približne algoritme među kojima se razlikuju heuristički i meta-heuristički. Riječi „meta" i „heuristika" potječu iz starogrčkog jezika sa značenjem „viša razina", odnosno „umjetnost otkrivanja novih metoda". Heuristički algoritmi odnose se na tehnike rješavanja problema i učenje temeljeno na iskustvu dok su meta-heuristike algoritamsko okruženje, neovisno o problemima, na visokoj razini koje pruža skup smjernica ili metoda za razvoj upravo heurističkih algoritama [56]. Pretraživanjem velikog skupa mogućih rješenja, meta-heurističke metode često daju vrlo dobra rješenja s manje računanja u odnosu na metode temeljene na računu (eng. *calculus-based methods*). Termin „meta-heuristike" prvi je upotrijebio Fred Glover 1986. godine.

Meta-heuristike mogu biti zasnovane na jednom rješenju ili na cijeloj populaciji rješenja. Prve se temelje na lokalnom pretraživanju prostora rješenja te su eksploatacijski orijentirane. Populacijske meta-heuristike orijentirane su na istraživanje, a karakterizira ih iterativno ažuriranje većeg broja mogućih rješenja koja se pamte sve dok uvjet prekida nije zadovoljen.

Meta-heuristike se koriste i za otkrivanje zlonamjernog softvera. Pregled takvih radova prikazan je u Tablici 3.1. U [33] autori su koristili genetsko programiranje kako bi izgradili model koji je razlučivao zlonamjerne od normalnih datoteka. Po uzoru na taj članak, otkrivanje zlonamjernog softvera provest će se i u ovom radu, ali uz korištenje meta-heurističkog pčelinjeg programiranja kojeg je Dervis Karaboga predstavio u [29]. U nastavku predstavljeni su osnovni koncepti pčelinjeg algoritma i pčelinjeg programiranja.

Članak	Korištene meta-heuristike	Godina
A malware detection model based on a negative selection algorithm with penalty factor [68]	Algoritam umjetnog imunološkog sustava (AIS)	2010
New malware detection system using metric-based method and hybrid genetic algorithm [30]	Genetski algoritam	2012
Malware detection by pruning of parallel ensembles using harmony search [50]	Harmonijski algoritam	2013
Virus detection using clonal selection algorithm with Genetic algorithm (VDC algorithm) [5]	Algoritam umjetnog imunološkog sustava (AIS)	2013
Malware Detection Using Genetic Programming [33]	Genetski algoritam	2014

Tablica 3.1: Pregled nekih korištenih meta-heurističkih pristupa pri otkrivanju zlonamjernog softvera [45].

3.1 Pčelinji algoritam

Iako svaka pčela u prirodi odrađuje samo jedan zadatak, kroz razne načine komunikacije, poput plesa u ritmu (eng. *waggle dance*), cijela kolonija uspijeva obavljati složene zadatke kao što su izgradnja košnice i sakupljanje pelina. U svakoj košnici nalaze se tri vrste pčela:

- **zaposlene pčele** (eng. *employed bees*)
  Njihov zadatak je iskorištavati izvor hrane s kojeg donose nektar u košnicu. Plesom prenose informacije o izvoru hrane pčelama promatračima.
- **pčele promatrači** (eng. *onlookers*)
  Nakon što iz plesa dobiju informacije o lokacijama izvora hrane, pčele promatrači odlaze iz košnice kako bi pretražili susjedstvo istih s ciljem pronalaska boljih izvora hrane.
- **pčele izviđači** (eng. *scout bees*)
  Bez ikakvog prethodnog znanja, zadatak pčela izviđača je nasumice pronaći nove izvore hrane u blizini košnice.

Motiviran prethodno opisanim inteligentnim ponašanjem pčela, pčelinji algoritam (eng. *artificial bee colony*, kratko ABC) je meta-heuristika koju je 2005. godine u [28] predstavio Dervis Karaboga. Intuitivan je poput algoritma roja čestice (eng. *particle swarm optimization*, kratko PSO)[15] i algoritma diferencijalne evolucije (eng. *differential evolution*, kratko DE)[16], a najjednostavnije varijante algoritma koriste samo uobičajene kontrolne parametre poput veličine populacije (N) i maksimalnog broja iteracija. Detaljni pseudokod osnovnog pčelinjeg algoritma prikazan je u Algoritmu 1.

[15] Populacijski meta-heuristički algoritam inspiriran kretanjem čestica u roju. Kretanje čestica, odnosno rješenja u prostoru pretraživanja, određeno je njihovom najboljom poznatom pozicijom kao i najboljom poznatom pozicijom cijele populacije rješenja.

[16] Algoritam diferencijalne evolucije upotrebljava evolucijske operatore kao što su križanje, mutacija i selekcija. Prilikom kreiranja boljih rješenja, koristi se križanjem i mutacijom. Operatori selekcije koriste se za usmjeravanja pretraživanja prostora ka potencijalnim rješenjima.

Pčelinji algoritam je optimizacijski alat čiji je prostor rješenja predstavljen terminom izvora hrane. Njega iterativno iscrpljuju umjetne pčele s ciljem pronalaska izvora s velikom količinom nektara te, u konačnici, izvora s najviše nektara.

Algoritam 1: Pseudokod osnovnog pčelinjeg algoritma [29].

Rezultat: Najbolje pronađeno rješenje

Inicijalizacija početne populacije X_i, $i = 1 \ldots N$.
Evaluacija inicijalne populacije.
Postavi broj iteracija na 1.

while *broj iteracija < maksimalni broj iteracija* **do**
  for *svaka zaposlena pčela* **do**
    Stvori novo rješenje v_i koristeći (3.2).
    Izračunaj vrijednost fit_i.
    Primijeni pohlepni postupak selekcije.
  Izračunaj vjerojatnosne vrijednosti p_i za rješenja (x_i) koristeći (3.3).
  for *svaka pčela promatrač* **do**
    Odaberi rješenje x_i na temelju vjerojatnosti p_i.
    Stvori novo rješenje v_i.
    Izračunaj vrijednost fit_i.
    Primijeni pohlepni postupak selekcije.
  if *postoji napušteno rješenje* **then**
    pčela izviđač ga zamijeni s novim nasumično kreiranim rješenjem koristeći (3.1).
  Zapamti najbolje rješenje do sada.
  Povećaj broj iteracija za jedan.

Populacija pčelinjeg algoritma sastoji se od N mogućih rješenja x_i koja predstavljaju izvore hrane. Prilikom inicijalizacije u Algoritmu 1, početnu populaciju nasumično generiraju umjetne pčele izviđači koristeći:

$$x_{ij} = x^j_{min} + \text{rand}(0,1)(x^j_{max} - x^j_{min}) \tag{3.1}$$

gdje su x^j_{min} i x^j_{max} donja i gornja granica j-tog parametra rješenja i.
Nakon inicijalizacije i evaluacije svih rješenja, započinje faza zaposlenih pčela u kojoj se stvaraju nova rješenja koristeći:

$$v_{ij} = x_{ij} + \phi_{ij}(x_{ij} - x_{kj}). \tag{3.2}$$

Pri čemu je ϕ_{ij} nasumičan broj između -1 i 1, v_i potencijalna pozicija izvora hrane, x_i trenutni izvor hrane, x_k susjedni izvor hrane, $j \in \{1, 2, \ldots, D\}$ nasumično odabrani indeks koji predstavlja komponentu svake pozicije izvora hrane, D dimenzija problema. Tako dobiveni v_i koriste se za usporedbu s x_i te, ukoliko je bolji izvor hrane, pohlepno zamjenjuje x_i u populaciji.

Nakon što sve zaposlene pčele završe s pretraživanjem prostora, odlaze u košnicu kako bi podijelile nove informacije s pčelama promatračima. Time započinje njihova faza u algoritmu. Svaka pčela promatrač izabire izvore hrane s obzirom na njihove potencijale p_i. Oni se računaju vjerojatnosno:

$$p_i = \frac{\alpha \times fit_i}{fit_{best}} + (1 - \alpha) \tag{3.3}$$

gdje je α parametar između 0 i 1, fit_{best} kvaliteta najboljeg poznatog rješenja (eng. *fitness value*)[17], a fit_i kvaliteta rješenja x_i proporcionalna količini nektara, tj. funkciji cilja f_i.

$$fit_i = \frac{1}{1 + f_i}. \tag{3.4}$$

U osnovnom pčelinjem algoritmu, pčela promatrač odabire rješenje rulet metodom[18] koja osigurava odabir rješenja s boljom funkcijom cilja. Nakon što odabere izvor hrane, pčela pretražuje njegovo susjedstvo s ciljem pronalaska novog izvora koristeći (3.2) kojeg onda uspoređuje s početnim.

Ukoliko je nektar na postojećim izvorima hrane istrošen, tj. ako se radi o napuštenom rješenju, na prethodne dvije faze nadovezuje se rad pčela izviđača koje nasumice pronalaze nove izvore koristeći (3.1).

Upravo ovom kombinacijom različitih zadataka koje obavljaju različite umjetne pčele, meta-heuristički pčelinji algoritam kombinira metode lokalnog pretraživanja prostora rješenja s globalnim s ciljem izbjegavanja zaglavljivanja u lokalnom optimumu.

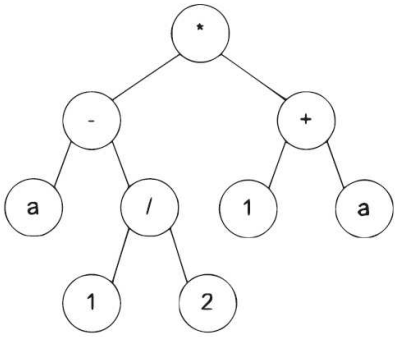
[17] *Fitness* funkcija je posebna vrsta funkcije cilja koja vrednuje optimalnost rješenja, odnosno kromosoma u evolucijskim algoritmima.

[18] Rulet metoda (eng. *roulette wheel selection*) selekcije je operator kod kojega se svakoj jedinki iz populacije koja se promatra izračuna vrijednost funkcije cilja te vjerojatnost p_i da ta jedinka bude odabrana.

3.2 Pčelinje programiranje

Prvi puta predstavljeno 2012. godine u [29], pčelinje programiranje (eng. *artificial bee colony programming*, kratko ABCP) je vrsta algoritma automatskog programiranja[19] utemeljena na evolucijskom računanju[20]. Štoviše, ono je nadogradnja prethodno opisanog pčelinjega algoritma po uzoru na genetsko programiranje[21]. Korištenjem pčelinjeg programiranja mogu se riješiti mnogi problemi među kojima je i simbolička regresija[22].

U algoritmima pčelinjeg programiranja izvori hrane predstavljaju matematičke modele sastavljene od terminala i funkcija zbog čega svako rješenje predstavlja jedan mogući model za promatrani sistem. Kao u genetskom programiranju, modeli su reprezentirani stablima čiji čvorovi predstavljaju terminale i funkcije. Listovi se uvijek označavaju terminalima, dok svi ostali čvorovi simboliziraju funkcijske vrijednosti s ciljem definiranja povezanosti u stablu. Ovisno o vrsti problema, funkcije i terminali mogu poprimiti različite vrijednosti, no najčešće se radi o konstantama i ulaznim varijablama u slučaju terminala te o trigonometrijskim i logaritamskim funkcijama, aritmetičkim i logičkim operatorima u slučaju funkcija. Primjer jednog takvog stabla nalazi se na Slici 3.1 gdje je a nezavisna, a $f(a) = (a - (\frac{1}{2})) * (1 + a)$ zavisna varijabla.



Slika 3.1: Primjer modela prikazanog stablom [29].

[19] Automatsko programiranje (eng. *automatic programming*) je sinteza programa iz specifikacije [27].

[20] Tehnike evolucijskog računanja (eng. *evolutionary computation*, kratko EC) su stohastički algoritmi čije su metode pretraživanja prirodno inspirirane [56].

[21] Genetsko programiranje (eng. *genetic programming*, kratko GP) je varijanta genetskih algoritama koju je prvi upotrijebio Koza 1992. godine. Kromosomi genetskog programiranja reprezentirani su hijerarhijskim stablima s promjenjivom duljinom [19].

[22] Simbolička regresija (eng. *symbolic regression*) je postupak oblikovanja matematičkog modela pomoću postojećeg konačnog uzorka vrijednosti nezavisnih varijabli i pridruženih vrijednosti zavisnih varijabli [29]. Odnosno, simboličkom regresijom pokušava se definirati matematička funkcija koja opisuje odnos između zavisnih i nezavisnih varijabli.

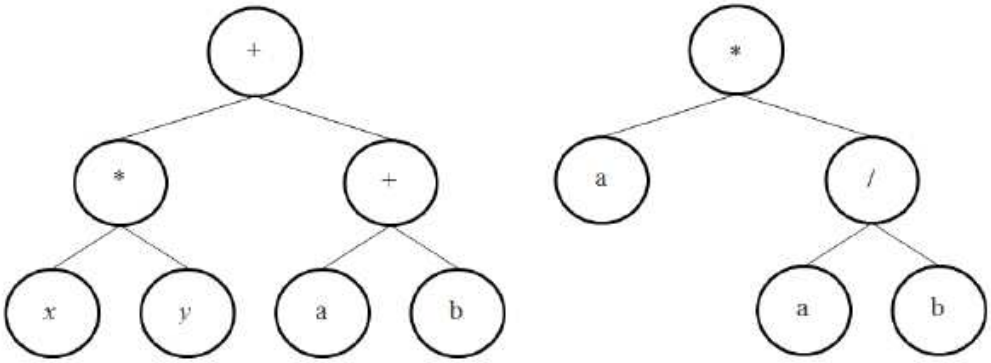
U svakoj instanci algoritma pčelinjeg programiranja potrebno je definirati terminale i funkcije. Primjerice, u [33] vrijednosti terminala su ulazne značajke potrebne za otkrivanje zlonamjernog softvera, dok su `+, -, *, /, sin, cos, exp, iff, log` neke moguće vrijednosti funkcija.

Kvaliteta svakog izvora hrane određuje se računanjem točnosti svakog od modela, pri čemu se koristi sirova funkcija cilja (eng. *raw fitness*). Ona predstavlja zbroj apsolutnih grešaka između rezultata dobivene i ciljane funkcije, tj. radi se o formuli

$$f(x_i) = \sum_{j=1}^{M} |g_j - t_j| \tag{3.5}$$

gdje je M ukupan broj slučajeva, g_j rezultat dobiven i-tim modelom, a t_j ciljani rezultat za j-ti slučaj.

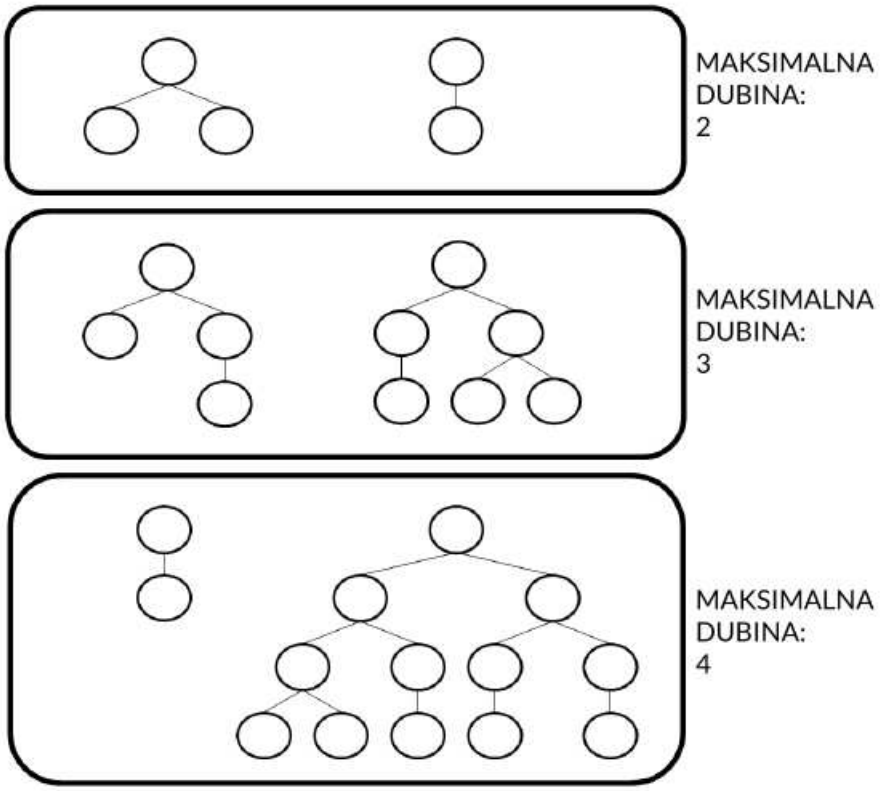
Pseudokod osnovnog pčelinjeg programiranja prikazan je u Algoritmu 2. On započinje inicijalizacijom početnih modela *ramped* pola-pola metodom. Odnosno, pola populacije generira se korištenjem metode rasta[23], a druga polovica korištenjem pune metode[24]. Na taj način osigurava se raznolikost što se može vidjeti i na Slici 3.3. Primjeri pune metode i metode rasta respektivno su prikazani na Slici 3.2.



Slika 3.2: Stablo generirano punom metodom (lijevo) i stablo generirano metodom rasta (desno) [20].

[23]Metoda rasta (eng. *grow method*) generira modele različith veličina i oblika.

[24]Puna metoda (eng. *full method*) generira modele, tj. stabla čiji su listovi maksimalno udaljeni od korijena. List je maksimalno udaljen od korijena stabla ukoliko je broj čvorova između njih jednak prethodno definiranoj maksimalnoj dubini stabla.

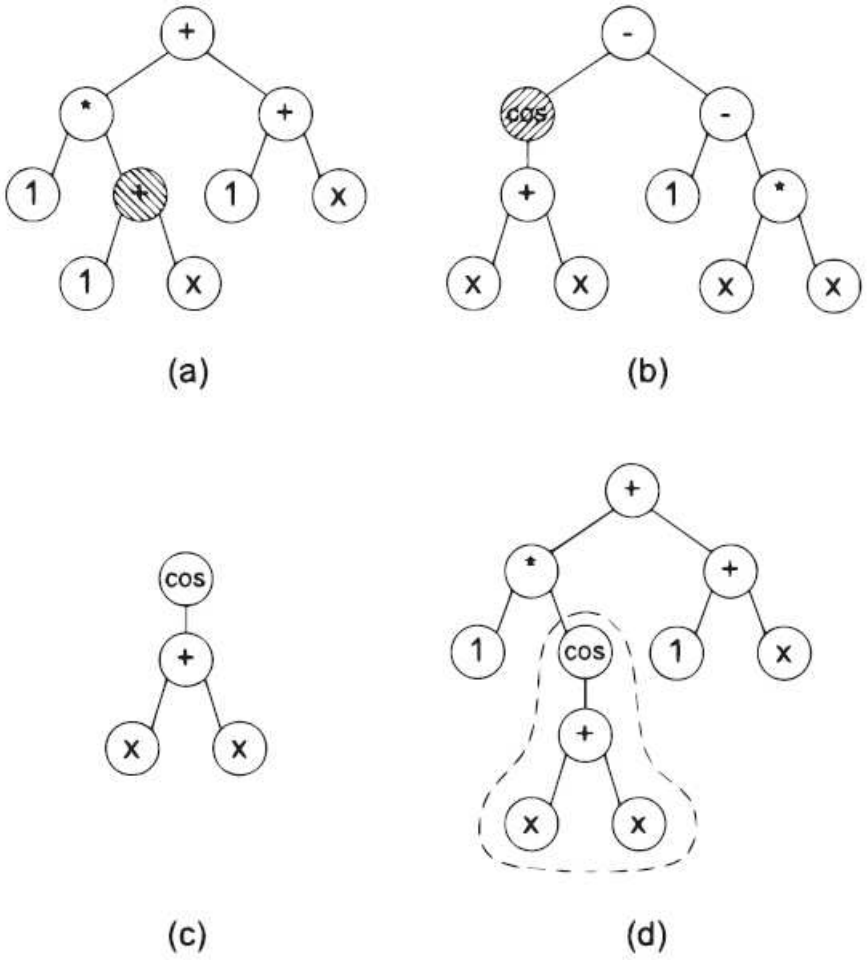


Slika 3.3: Ilustracija generiranja inicijalne populacije korištenjem *ramped* pola-pola metode u ovisnosti o dopuštenoj maksimalnoj dubini stabla [20].

Nakon generiranja početnih modela radi se njihova evaluacija, uoči čega započinje faza zaposlenih pčela. Ona, kao i faza pčela promatrača i izviđača, provodi se iterativno na sličan način kao i u pčelinjem algoritmu uz bitnu razliku načina stvaranja novih rješenja. Kako je i napisano u Algoritmu 2, ono se provodi korištenjem **mehanizma za razmjenu podataka** (eng. *information sharing mechanism*). Jedan jednostavan primjer razmjene podataka prikazan je na Slici 3.4. Mehanizam je osmišljen kao adaptacija metode križanja[25] genetskog programiranja.

[25]Operator križanja je stohastički operator varijacije s ciljem stvaranja novih jedinki kombiniranjem nekoliko starih jedinki.

Dakle, kako bi se pretražilo susjedstvo rješenja x_i, nasumice se odabere neko drugo rješenje x_k ($k \neq i$) iz populacije koje će zajedno s x_i imati ulogu roditelja u procesu križanja. Prvi korak križanja je odabrati čvor u x_k. On može biti funkcija (vjerojatnost: P_{ip}) ili terminal (vjerojatnost: $(1 - P_{ip})$). Na sličan način odabire se i čvor u roditelju x_i. Konačno, susjed od x_i nastaje zamjenom podstabla odabranog čvora u x_i s podstablom odabranog čvora u x_k u početnom rješenju x_i.



Slika 3.4: Primjer mehanizma razmjene podataka pri čemu **(a)** predstavlja prvog roditelja x_i, **(b)** drugog roditelja x_k, **(c)** podstablo odabrano iz x_k, **(d)** novonastalog susjeda od x_i [24].

Algoritam 2: Pseudokod osnovnog pčelinjeg programiranja [29].

Rezultat: Najbolje pronađeno rješenje.

Inicijalizacija početnih rješenja X_i *ramped* metodom.
Evaluacija inicijalnih rješenja.
Postavi broj iteracija na 1.

while *broj iteracija* < *maksimalni broj iteracija* **do**
  for *svaka zaposlena pčela* **do**
    Stvori novo rješenje v_i koristeći mehanizam za razmjenu podataka.
    Izračunaj točnost rješenja koristeći (3.4) i (3.5).
    Primijeni pohlepni postupak selekcije između x_i i v_i.
  Izračunaj vjerojatnosne vrijednosti p_i za rješenja koristeći (3.3).
  for *svaka pčela promatrač* **do**
    Odaberi rješenje x_i na temelju vjerojatnosti p_i.
    Stvori novo rješenje v_i koristeći mehanizam za razmjenu podataka.
    Izračunaj točnost rješenja v_i koristeći (3.4) i (3.5).
    Primijeni pohlepni postupak selekcije između x_i i v_i.
  if *postoji napušteno rješenje* **then**
    pčela izviđač ga zamijeni s novim nasumično generiranim rješenjem koristeći (3.1).
  Zapamti najbolje rješenje do sada.
  Povećaj broj iteracija za jedan.

3.2.1 Poboljšano pčelinje programiranje

Standardno pčelinje programiranje ima nekoliko nedostataka među kojima su i usporena konvergencija pri pronalasku lokalnog minimuma te niska lokalnost [26]. Kako bi poboljšao njegovu učinkovitost, Dervis Karaboga 2019. godine u [24] predstavio je tri nove verzije Algoritma 2. Brzo pčelinje programiranje (eng. *qABCP*, kratko od *quick artificial bee colony programming*) jedna je od tih verzija koja unapređuje konvergenciju algoritma modifikacijom **što** algoritam radi. Druga verzija je tzv. semantičko pčelinje programiranje (eng. *sABCP*, kratko od *semantic artificial bee colony programming*) koje modifikacijom **kako** algoritam obavlja mehanizam razmjene podataka povećava njegovu lokalnost. Posljednja verzija kombinacija je prethodne dvije (eng. *qsABCP*).

[26] U evolucijskom računanju, lokalnost je svojstvo kada susjedni genotipi odgovaraju susjednim fenotipima. Klasificira se kao visoko i nisko. Algoritmi koji kreiraju populacije s visokom lokalnošću imaju bolji učinak u pronalasku najboljeg rješenja, što se pokazalo i u [22].

3.2.1.1 qABCP

U klasičnom pčelinjem programiranju, pčele promatrači nakon odabira izvora hrane pokušavaju direktno unaprijediti isti. Suprotno tome, kod brzog pčelinjeg programiranja odabrani izvor hrane pospješava se odlaskom pčela promatrača u njegovu okolinu, pronalaska najboljeg izvora u toj okolini, odnosno susjedstvu, te unaprijeđenju upravo tog izvora korištenjem mehanizma za razmjenu podataka. Odnosno, ova varijanta Algoritma 2 temelji se na izmjeni načina rada pčela promatrača, što rezultira pronalazak kvalitetnijih rješenja u kraćem broju iteracija. Pseudokod brzog pčelinjeg programiranja nalazi se u Algoritmu 3.

Algoritam 3: Pseudokod brzog pčelinjeg programiranja [24].

Rezultat: Najbolje pronađeno rješenje.

Inicijalizacija početnih rješenja X_i *ramped* metodom.
Evaluacija inicijalnih rješenja.
Postavi broj iteracija na 1.

while *broj iteracija* < *maksimalni broj iteracija* **do**
- **for** *svaka zaposlena pčela* **do**
  - Stvori novo rješenje v_i koristeći mehanizam za razmjenu podataka.
  - Izračunaj točnost rješenja koristeći (3.4) i (3.5).
  - Primijeni pohlepni postupak selekcije između x_i i v_i.
- Izračunaj vjerojatnosne vrijednosti p_i za rješenja koristeći (3.3).
- **for** *svaka pčela promatrač* **do**
  - Odaberi rješenja x_i na temelju vjerojatnosti p_i.
  - Pronađi najbolje rješenje $x_{N_i}^{best}$ u susjedstvu rješenja x_i.
  - Stvori novo rješenje $v_{N_i}^{best}$ iz $x_{N_i}^{best}$ koristeći mehanizam za razmjenu podataka.
  - Izračunaj točnost rješenja $v_{N_i}^{best}$ koristeći (3.4) i (3.5).
  - Primijeni pohlepni postupak selekcije između $x_{N_i}^{best}$ i $v_{N_i}^{best}$.
- **if** *postoji napušteno rješenje* **then**
  - pčela izviđač ga zamijeni s novim nasumično generiranim rješenjem koristeći (3.1).
- Zapamti najbolje rješenje do sada.
- Povećaj broj iteracija za jedan.

Kako bi Algoritam 3 bio u potpunosti jasan, definirano je značenje *susjedstva* rješenja, odnosno mjera sličnosti prikladna za problem simboličke regresije:

$$d(i,m) = \frac{\sum_{t=1}^{T} \mid F_{x_i}(f_{c_t}) - F_{x_m}(f_{c_t}) \mid}{T}. \tag{3.6}$$

Pri tome je $d(i,m)$ iznos funkcije udaljenosti između rješenja x_i i x_m, T broj primjera pomoću kojih treniramo rješenje, a F_{x_i} i F_{x_m} funkcije koje predstavljaju rješenje x_i, odnosno x_m. Prosječna udaljenost svih rješenja u populaciji do rješenja x_i dana je formulom:

$$md_i = \frac{\sum_{m=1}^{N} d(i,m)}{N-1} \tag{3.7}$$

gdje je N ukupan broj rješenja u populaciji. Okolina rješenja x_i u kojoj se nalaze svi njegovi susjedi dobiva se umnoškom koeficijenta radijusa susjedstva $r \geq 0$ i prosječne udaljenosti svih rješenja od x_i. Konačno, susjedstvo rješenja x_i određuje se računanjem udaljenosti svakog rješenja u populaciji od x_i te utvrđivanjem pripada li ono u kružnu okolinu oko x_i:

$$x_m = \begin{cases} \text{u susjedstvu od } x_i & \textbf{ako } d(i,m) \leq r \times md_i \\ \text{nije u susjedstvu od } x_i & \textbf{inače} \end{cases}. \tag{3.8}$$

3.2.1.2 sABCP

„Mala promjena u genotipu[27] rješenja dovodi do male promjene u njegovom fenotipu[28]" je misao vodilja termina lokalnosti. Odnosno, ona je ključna jer u mehanizmu za razmjenu podataka na ovaj način nastaju nova, ali vrlo slična rješenja u odnosu na genotip i fenotip. Kako bi povećali lokalnost u algoritmima pretraživanja prostora, stručnjaci promatraju sintaksu i semantiku rješenja. Kako se semantičko poboljšavanje lokalnosti pokazalo uspješnije u genetičkom programiranju, Dervis Karaboga je predložio semantičko pčelinje programiranje koje se, u usporedbi s Algoritmom 2, razlikuje po mehanizmu razmjene podataka. Kako bi se izbjeglo ručno računanje semantičke osjetljivosti rješenja, ali i dalje zadržala mala semantička promjena kod djece nakon križanja, predlaže se korištenje strukture **uglavnom semantički sličnog križanja** (eng. *MSSC*, kratko od *most semantically similar crossover*) u mehanizmu razmjene podataka čiji pseudokod je prikazan u Algoritmu 4.

[27]Genotip označava sveukupno nasljeđe nekog organizma.

[28]Fenotip označava sve osobine nekog organizma, odnosno on je izraz genotipa.

Algoritam 4: Pseudokod mehanizma za izmjenu podataka korištenog u semantičkom pčelinjem programiranju [24].

Rezultat: Novo rješenje.

Nasumično odaberi rješenje x_k iz populacije tako da $i \neq k$.
MAX = Veliki broj.
Postavi broj iteracija na 0.

while *broj iteracija* < *maksimalni broj iteracija u mehanizmu* **do**
    Nasumično odaberi čvor u rješenju x_i i odredi podstablo ST_i.
    Nasumično odaberi čvor u rješenju x_k i odredi podstablo ST_k.
    Izračunaj udaljenost podstabala ST_i i ST_k koristeći 3.6.
    if $d(i,k) < MAX$ ***and*** $d(i,k) > LBSS$ **then**
        $MAX = d(i,k)$.
        Postavi ST_i za mjesto promjene u x_i.
        Postavi ST_k za mjesto promjene u x_k.
    Povećaj broj iteracija za jedan.
Postavi ST_k na mjesto ST_i u rješenju x_i.

U Algoritmu 4 pojavljuje se koeficijent $LBSS$ (eng. *lower bound for semantic sensitivity*) koji ograničava da odabrano podstablo drugog roditelja ne bude previše slično ili identično odabranom podstablu prvog roditelja.

3.2.1.3 qsABCP

Algoritam 5 prikazuje pseudokod brzog semantičkog pčelinjeg programiranja nastalog kombinacijom prethodne dvije verzije.

Algoritam 5: Pseudokod brzog semantičkog pčelinjeg programiranja [24].

Rezultat: Najbolje pronađeno rješenje.

Inicijalizacija početnih rješenja X_i *ramped* metodom.
Evaluacija inicijalnih rješenja.
Postavi broj iteracija na 1.

while *broj iteracija* < *maksimalni broj iteracija* **do**
- **for** *svaka zaposlena pčela* **do**
  - Stvori novo rješenje v_i koristeći mehanizam za razmjenu podataka prikazan u Algoritmu 4.
  - Izračunaj točnost rješenja koristeći (3.4) i (3.5).
  - Primijeni pohlepni postupak selekcije između x_i i v_i.
- Izračunaj vjerojatnosne vrijednosti p_i za rješenja koristeći (3.3).
- **for** *svaka pčela promatrač* **do**
  - Odaberi rješenje x_i na temelju vjerojatnosti p_i.
  - Pronađi najbolje rješenje $x_{N_i}^{best}$ u susjedstvu rješenja x_i.
  - Stvori novo rješenje $v_{N_i}^{best}$ iz $x_{N_i}^{best}$ koristeći mehanizam za razmjenu podataka prikazan u Algoritmu 4.
  - Izračunaj točnost rješenja $v_{N_i}^{best}$ koristeći (3.4) i (3.5).
  - Primijeni pohlepni postupak selekcije između $x_{N_i}^{best}$ i $v_{N_i}^{best}$.
- **if** *postoji napušteno rješenje* **then**
  - pčela izviđač ga zamijeni s novim nasumično generiranim rješenjem koristeći (3.1).
- Zapamti najbolje rješenje do sada.
- Povećaj broj iteracija za jedan.

Poglavlje 4

Implementacija i rezultati

Po uzoru na [33], gdje su autori uz pomoću genetskog programiranja izradili model koji je razlučivao zlonamjerne od normalnih datoteka, pretraga prostora svih takvih modela provest će se i u ovom radu, ali uz korištenje meta-heurističkog algoritma pčelinjeg programiranja koji je opisan u Poglavlju 3.2.

Prema našim saznanjima, pčelinji algoritam prethodno nije uporabljen za otkrivanje zlonamjernog softvera.

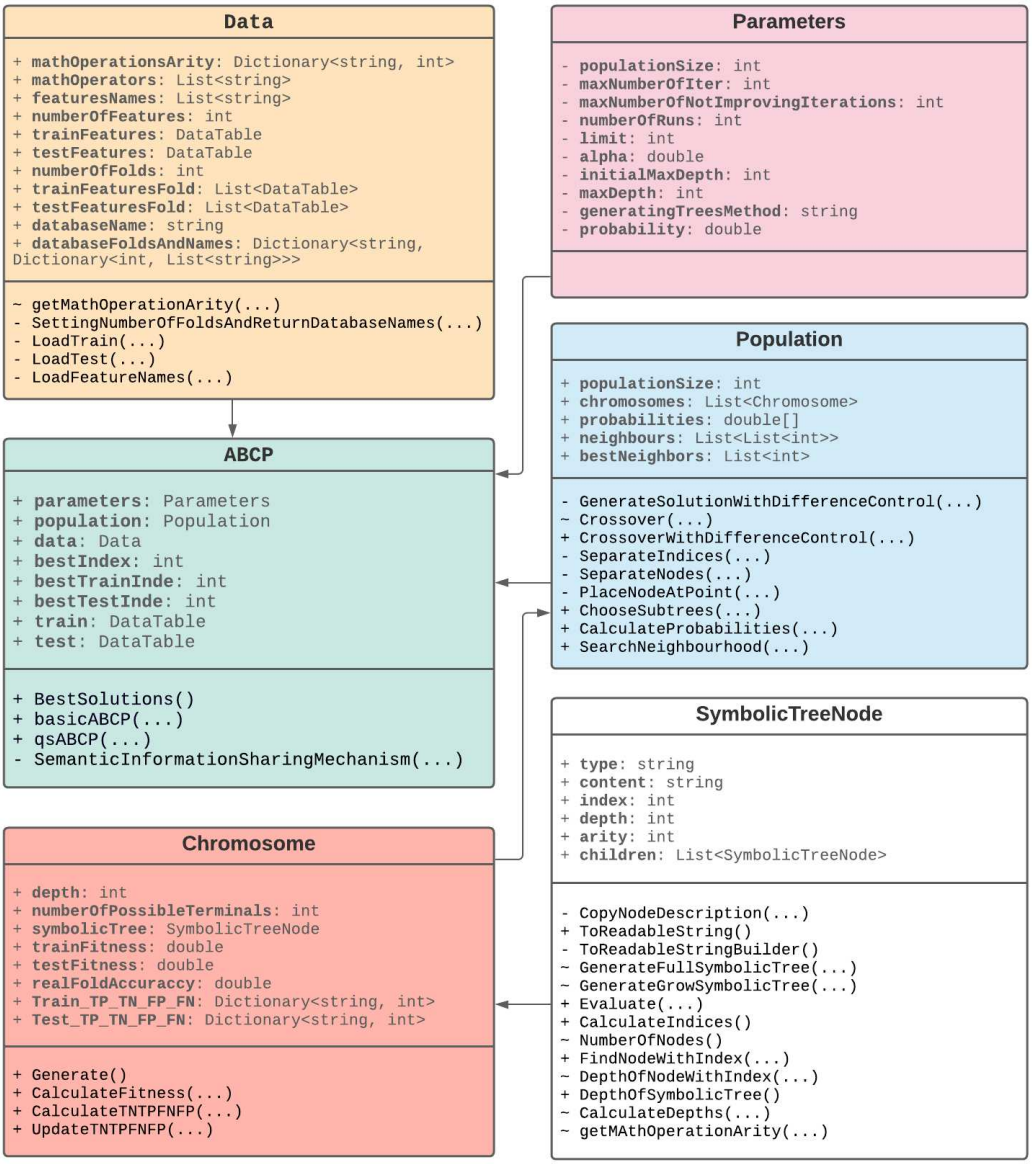
4.1 Implementacija

Za pretraživanje prostora modela za razlučivanje softvera pčelinjim programiranjem bilo je potrebno implementirati Algoritam 2 i 5 iz Poglavlja 3.2, te razviti korisničko sučelje što je i učinjeno u programskom jeziku `C#`. Izvorni kod može se pogledati na: `https://github.com/mateastanisic/antico`.

4.1.1 Pregled implementiranih klasa za potrebe pčelinjeg programiranja

Kako bi se što jasnije prikazala struktura dijela sučelja relevantnog za elemente pčelinjeg programiranja, na Slici 4.1 prikazan je dijagram klasa.

Jedna od potrebnih klasa je `Parameters` u kojoj su pohranjene sve vrijednosti odabranih parametara potrebnih za pokretanje pčelinjeg programiranja. Nadalje, uz pomoć klase `Data` postavljaju se ostale potrebne vrijednosti za kreiranje stabala, odnosno mogućih modela za otkrivanje zlonamjernog softvera. Kako bi se isti vrednovali, u istu klasu učitavaju se i podaci o datotekama iz baze podataka.



Slika 4.1: Dijagram korištenih klasa potrebnih za implementaciju pčelinjeg programiranja.

Svaki čvor stabla samo je jedna instanca klase `SymbolicTreeNode` s jedinstvenom oznakom (`index`) u tekućem stablu. Osim oznake, svaki čvor stabla definiran je i s njegovim tipom, sadržajem, brojem djece, djecom i dubinom položaja u stablu. Također, razne metode kao što su generiranje novog podstabla, odnosno stabla čiji je korijen tekući čvor i njegovo pretvaranje u korisniku čitljiv pisani oblik, pokrivene su u ovoj klasi.

Za svaku instancu klase `Chromosome` može se reći da predstavlja jedno moguće rješenje problema, odnosno jedan model. Svaki model ima korijen stabla koji ga reprezentira te njegovu dubinu. Radi optimizacije, svaka instanca ove klase pamti i tekuće vrijednosti *fitness* funkcije, ali i tekući broj točno i netočno otkrivenih datoteka, negativno i pozitivno.

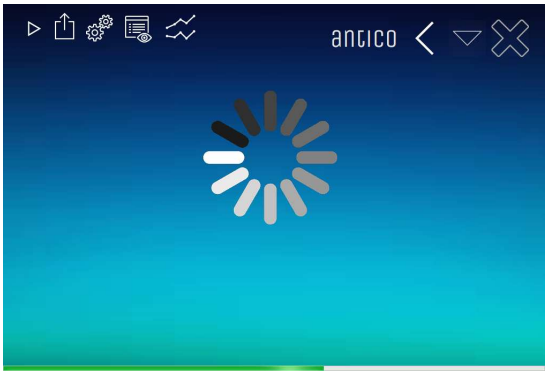
Kao i u svakom populacijskom algoritmu, i ova implementacija pčelinjeg programiranja sastoji se od klase `Population`. Naravno, svaka njena instanca je reprezentirana skupom mogućih rješenja, odnosno listom instanci klase `Chromosome`. Veličina te liste također je jedna od značajki klase (`populationSize`), a odabire ju korisnik prilikom definiranja parametara za pčelinje programiranje. Uz ove osnovne atribute, svaka populacija definirana je i listom susjeda svakog od rješenja te vjerojatnostima koje služe za odabir izvora hrane u fazi pčela promatrača. Za potrebe njihovog definiranja implementirane su metode `SearchNeighbourhood` i `CalculateProbabilities`, respektivno. Također, nad svaka dva rješenja u populaciji može se provesti križanje koristeći metodu `Crossover` uz pomoć metoda `SeparateIndices`, `SeparateNodes`, `PlaceNodeAtPoint` i `ChooseSubtrees`.

Konačno, nakon definiranih parametara i podataka potrebnih za generiranje populacije, u svakoj instanci klase `ABCP` moguće je pokrenuti pretraživanje prostora svih modela za otkrivanje zlonamjernog softvera koristeći osnovno (`BasicABCP`) ili brzo semantičko pčelinje programiranje (`qsABCP`). Za njihovo izvršavanje definirana je pomoćna metoda za pronalazak najboljeg rješenja u populaciji (`BestSolutions`), dok je Algoritam 4 implementiran u `SemanticInformationSharingMechanism`.

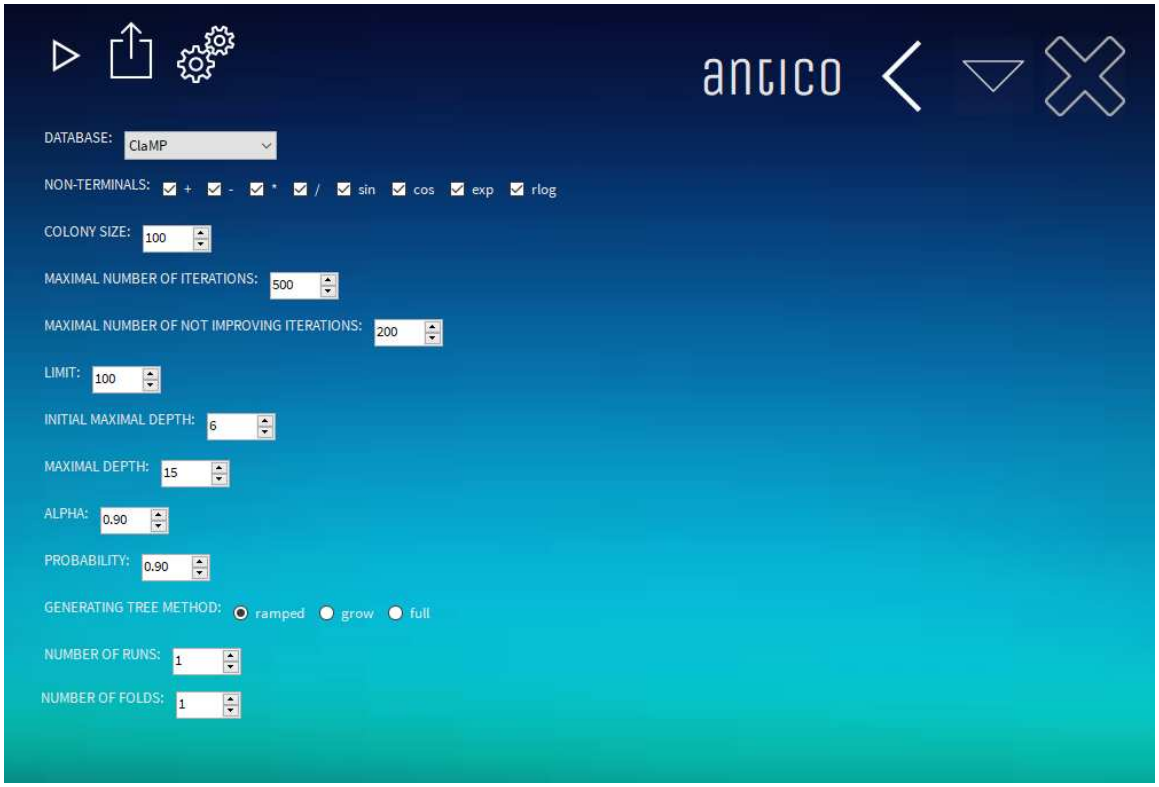
4.1.2 Pregled korisničkog sučelja

Na Slici 4.2 prikazano je početno sučelje s tri opcije za korisnika. Prva među njima je mogućnost kreiranja novog modela uz odabir parametara kao na Slici 4.4.

Slika 4.2: Početno sučelje.



Slika 4.3: Sučelje za vrijeme pretraživanja prostora modela.

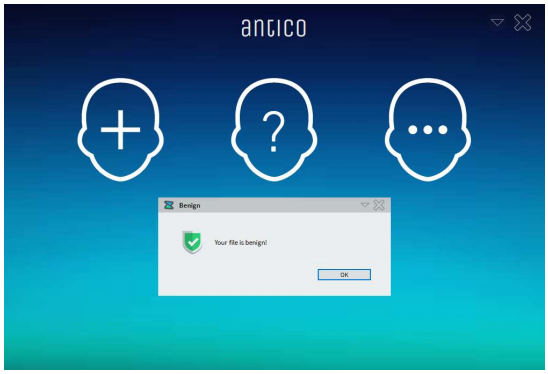


Slika 4.4: Odabir parametara.

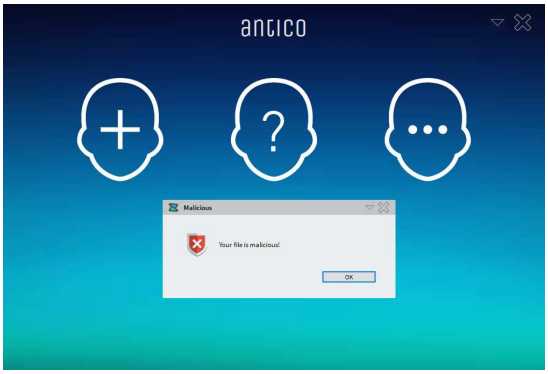
Nakon odabira parametara, pretražuje se prostor svih modela pri čemu sučelje izgleda kao na Slici 4.3. Za to vrijeme, korisnik ima opciju prikaza prozora s detaljnim ispisom algoritma (Slika 4.7), ali i prozora koji grafički prikazuje napredak *fitness* funkcije trenutno najboljeg rješenja u populaciji, njegove dubine ali i drugih vrijednosti kroz prijašnje iteracije algoritma.

Nakon završetka algoritma, sučelje će prikazati vrijednost *fitness* funkcije najboljeg pronađenog modela, ali i izgled njegovog stabla u čitljivom obliku. Moguće je vidjeti i njegov grafički prikaz. Naposljetku, korisnik može spremiti pronađeno rješenje na svoje računalo kojeg kasnije može i učitati kako bi ponovno vidio njegove značajke (Slika 4.8).

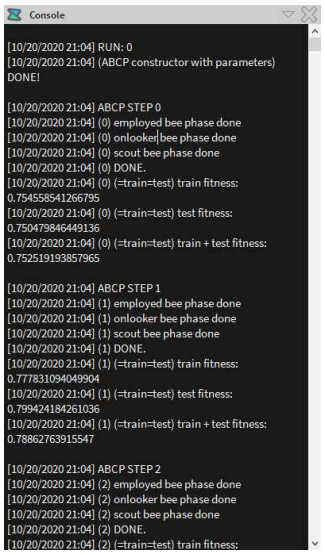
Povratkom na početno sučelje, korisnik može učitati datoteku za koju želi provjeriti je li zlonamjerna ili ne, te za istu dobiti odgovor kao na Slici 4.5 i 4.6. Posljednja opcija početnog sučelja pruža korisniku osnovne informacije o aplikaciji.



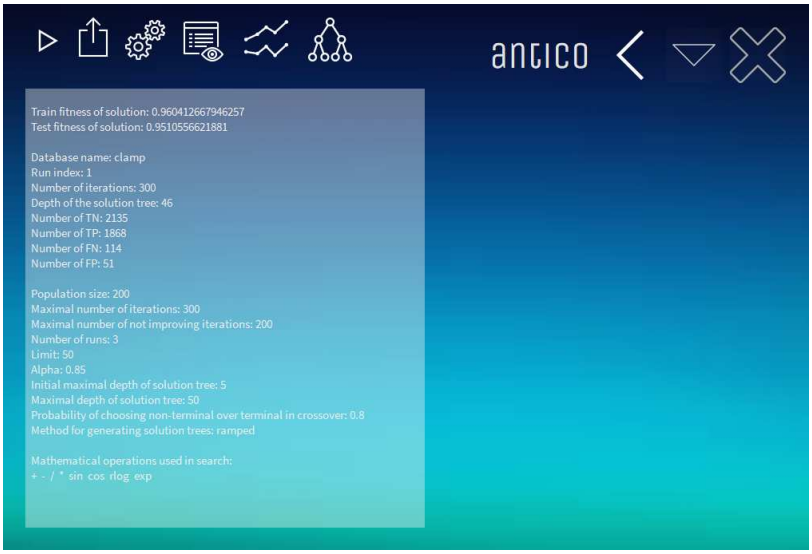
Slika 4.5: Obavijest korisniku da učitana datoteka nije zlonamjerna.



Slika 4.6: Obavijest korisniku da je učitana datoteka zlonamjerna.



Slika 4.7: Primjer prozora s detaljnim ispisom algoritma.



Slika 4.8: Sučelje nakon učitavanja prethodno spremljenog modela.

4.2 Rezultati

Kako bi se ispitale performanse Algoritma 2 i 5 za problem prepoznavanje zlonamjernog softvera, provedena su testiranja na četiri različita skupa podataka pri čemu su sva izvršena na računalu s Intel i5 frekvencije radnog takta 1.80GHz i 8GB radne memorije.

Značajnu funkciju prilikom testiranja imao je pomni odabir parametara pčelinjeg programiranja. Najbolji rezultati dobiveni u razumnom vremenu postizali su se za parametre

prikazane u Tablici 4.1. Svako povećanje parametra veličine populacije, maksimalnog broja iteracija, maksimalnog broja iteracija u kojima se najbolje rješenje nije poboljšalo ili maksimalne dubine stabla rezultiralo je produljenjem trajanja algoritma što ga ja činilo nepraktičnim.

Parametar	**Vrijednost**
Veličina populacije	100, 200
Maksimalan broj iteracija	200, 300, 400
Maksimalan broj iteracija u kojima se najbolje rješenje nije poboljšalo	200
Inicijalna maksimalna dubina stabla	5
Maksimalna dubina stabla	40, 50
Vjerojatnost odabira čvora u stablu s oznakom funkcije prilikom odabira podstabla drugog roditelja u mehanizmu za razmjenu podataka	0.9
Moguće funkcije	+, -, *, /, sin, cos, exp, log
α	0.9
Veliki broj (MAX) korišten u Algoritmu 4	1 000 000
Maksimalan broj iteracija u mehanizmu Algoritma 4	10
Donja granica semantičke osjetljivosti (LBSS)	0.01

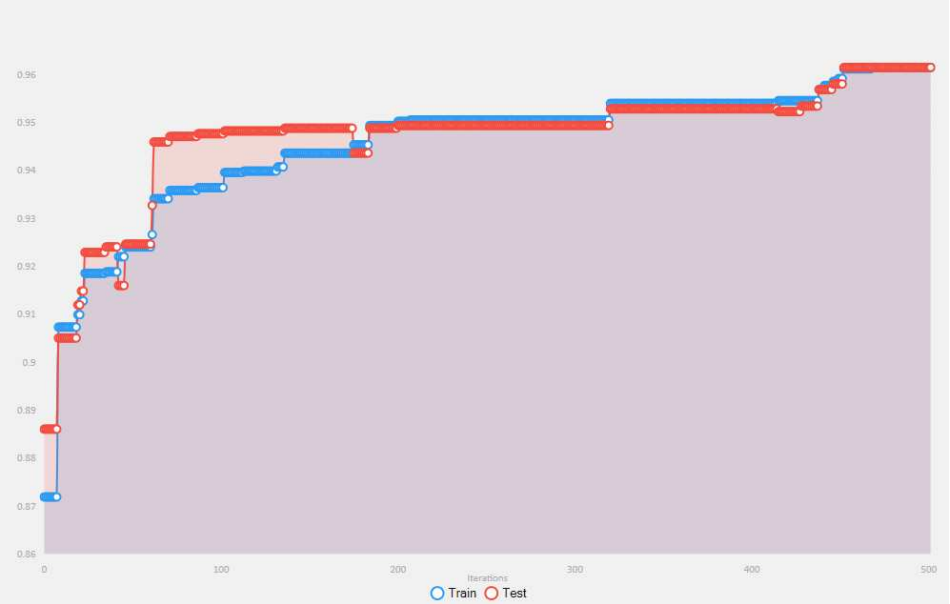
Tablica 4.1: Najčešće korišteni parametri pri testiranju.

Kako bi se odredila kvaliteta modela dobivenog pčelinjim programiranjem, promatra se vrijednost *fitness* funkcije nad testnim skupom podatka.

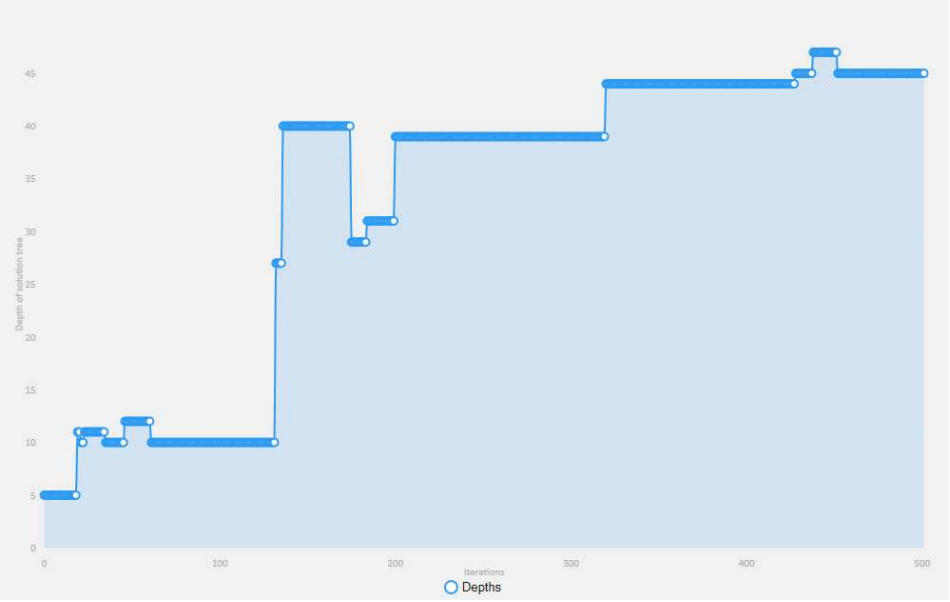
4.2.1 ClaMP

Prvi skup podataka nad kojim je provedeno testiranje je ClaMP (kratko od *classification of malware with PE headers*). Stvoren je 2016. godine iz 2722 zlonamjernih i 2488 benignih pakiranih datoteka [1]. Svaka datoteka opisana je s 69 značajki izrudarenih iz njihovih zaglavlja (*DOS header*, *File header*, *Optional header*).

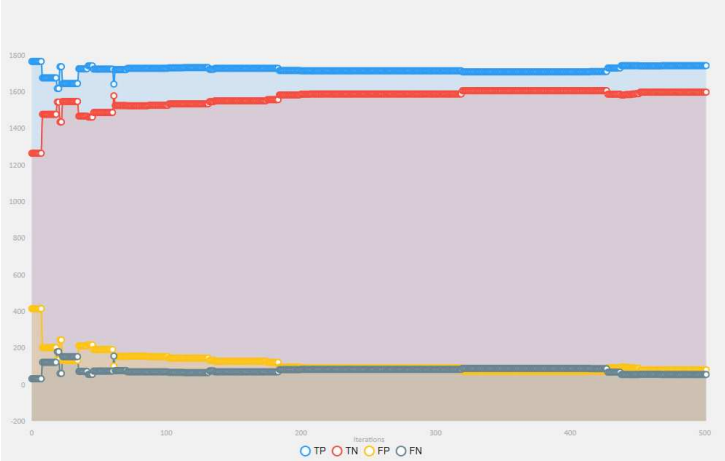
Na Slici 4.9 prikazan je napredak *fitness* funkcije najboljih modela za ClaMP podatke kroz iteracije Algoritma 2 koji je rezultirao globalno najboljim pronađenim modelom za te podatke. Za tu istu instancu algoritma na Slici 4.10 prikazan je napredak dubina stabla najboljih modela, a na Slici 4.11 napredak njihovih TP, TN, FP i FN vrijednosti.



Slika 4.9: Napredak *fitness* funkcije najboljeg modela za ClaMP podatke kroz iteracije Algoritma 2.



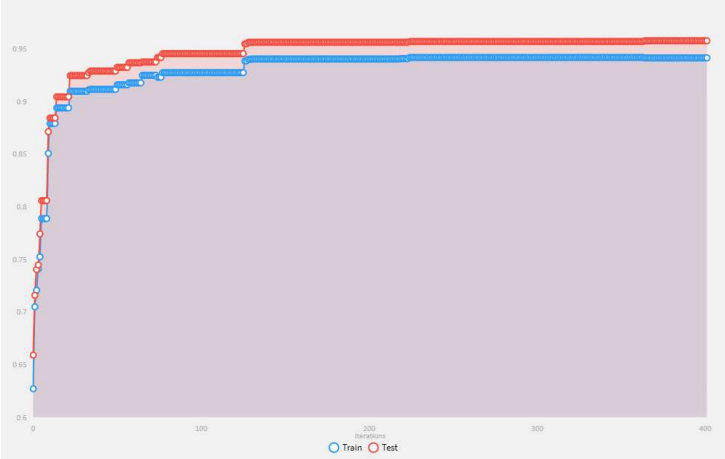
Slika 4.10: Napredak dubine stabla najboljeg modela za ClaMP podatke kroz iteracije Algoritma 2.



Slika 4.11: Napredak TP, TN, FP i FN vrijednosti najboljeg modela za ClaMP podatke kroz iteracije Algoritma 2.

Odnosno, najbolji pronađeni model korištenjem pčelinjeg programiranja za ClaMP podatke ima točnost od 96.14 % na testnom skupu podataka. U usporedbi s rezultatima iz [47] gdje su autori usporedili točnost različitih klasifikatora strojnog učenja nad istim skupom podataka, odabir pčelinjeg algoritma pokazao se kao konkurentan izbor. U Tablici 4.2 može se vidjeti njihova usporedba.

Nad istim podacima i sličnim parametrima testiran je i Algoritam 5. Iako se njegovim pokretanjem vrlo brzo pronašao dobar model što se može vidjeti i na Slici 4.12, njegova složenost učinila ga je nepraktičnim za izvođenje na korištenom računalu.



Slika 4.12: Napredak *fitness* funkcije modela za ClaMP podatke kroz iteracije Algoritma 5.

Model	Točnost[29]	Preciznost[30]	Odziv[31]	F1-mjera[32]
Naivni Bayes	57	0.732	0.204	0.3370
Logistička regresija	79.5	0.756	0.911	0.826
K najbližih susjeda	90.7	0.913	0.914	0.914
Stablo odluke	95.4	0.978	0.967	0.972
Nasumične šume	97.1	**0.992**	0.988	**0.99**
Metoda potpornih vektora	60	0.981	0.255	0.406
XGBoost	**97.47**	0.989	**0.99**	**0.99**
Višeslojni perceptron	88.9	0.878	0.923	0.89
Pčelinje programiranje	**96.14**	**0.957**	**0.971**	**0.964**

Tablica 4.2: Usporedba kvalitete različitih klasifikatora nad ClaMP podacima bez prethodnog odabira podskupa značajki.

4.2.2 *Angelio*

Kako bi se ispitale performanse pčelinjeg programiranja na nebalansiranim podacima, provedeno je testiranje nad podacima simbolično nazvanih *Angelio* po njihovom autoru [39]. Radi se o 42797 primjeraka zlonamjernog i 1079 benignog softvera, pri čemu su svi opisani s 1000 značajki dobivenih statičkom analizom pakiranih zlonamjernih datoteka preuzetih s `virusshare.com` te pakiranih benignih datoteka preuzetih s `portableapps.com` i *Windows 7 x86* direktorija.

Kako bi se smanjila složenost algoritma, broj značajki je prethodno reduciran na 648 korištenjem rekurzivne metode eliminacije značajki s 3-strukom unakrsnom validacijom i korištenjem nasumičnih šuma.

Koristeći ove podatke, pčelinjim programiranjem pronađen je model točnosti 97.09 % na testnom skupu podataka. Napredak *fitness* funkcije najboljih modela kroz iteracije tog pokretanja algoritma može se vidjeti na Slici 4.13, dok su u Tablici 4.3 uspoređeni rezultati tog modela s rezultatom dobivenim dubokim učenjem nad grafovima konvolucijskih neuronskih mreža (eng. *deep graph convolutional neural networks*, kratko *DGCNN*) [40].

---

[29] Točnost (eng. *accuracy*) je udio točno klasificiranih primjera u skupu svih primjera. $\frac{TP+TN}{TP+TN+FP+FN}$.

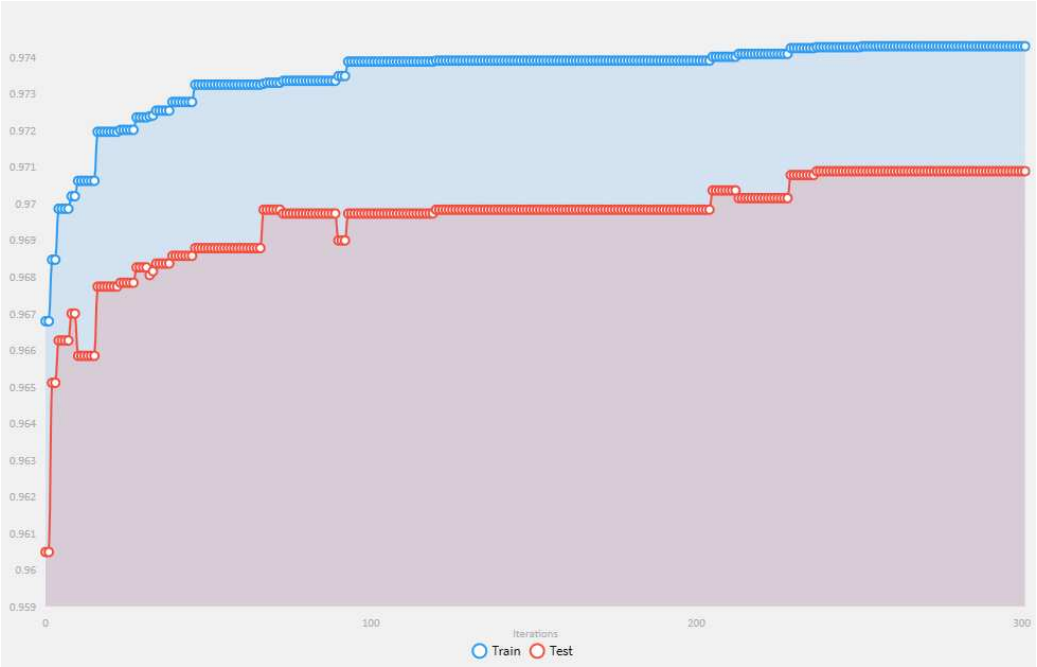
[30] Preciznost (eng. *precision*) je udio točno klasificiranih primjera u skupu pozitivno klasificiranih primjera. $PPV = \frac{TP}{TP+FP}$.

[31] Odziv (eng. *recall*) je udio točno klasificiranih primjera u skupu svih prepoznatih primjera. $TPR = \frac{TP}{TP+FN}$.

[32] F1-mjera je harmonijska sredina preciznosti i odziva. $F_1 = 2 \cdot \frac{PPV \cdot TPR}{PPV+TPR}$.

Model	F1-mjera
DGCNN s jednim slojem	0.9941
DGCNN s dva sloja	0.9942
Mreža dugotrajnog kratkoročnog pamćenja (LSTM)	**0.9953**
Pčelinje programiranje	**0.9849**

Tablica 4.3: Usporedba F1-mjere različitih klasifikatora nad *Angelio* podacima.



Slika 4.13: Napredak *fitness* funkcije najboljeg modela za *Angelio* podatke kroz iteracije Algoritma 2.

4.2.3 *Tek*

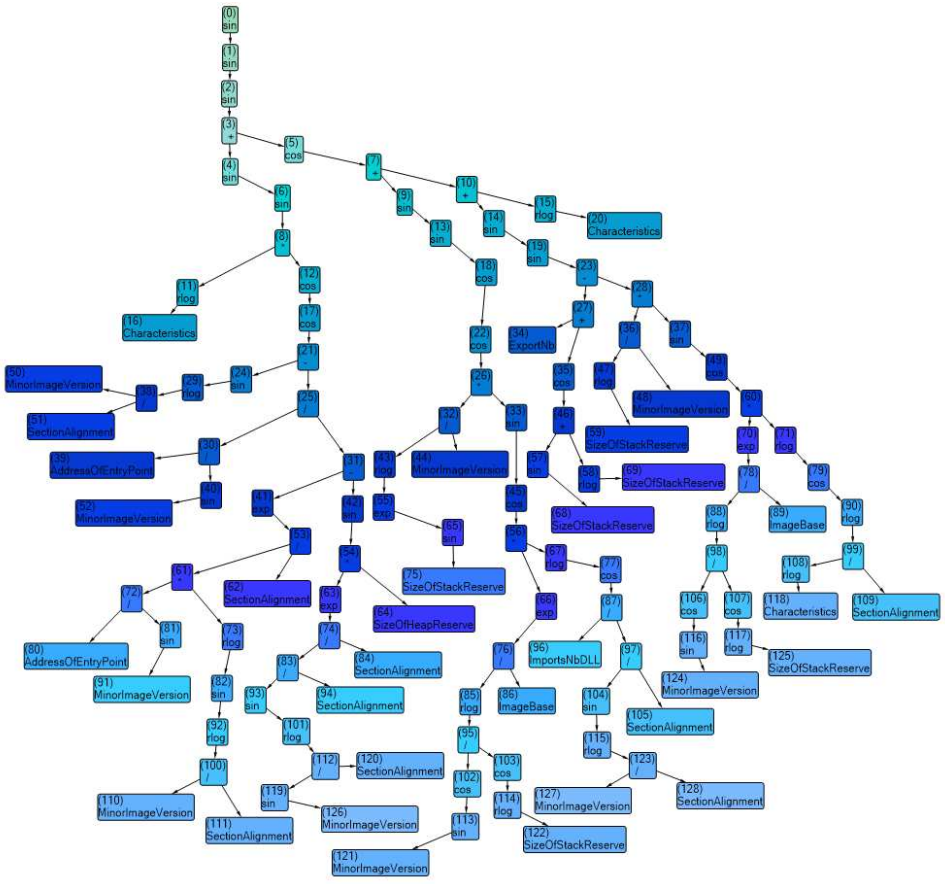
Tek baza podataka [2], simbolično nazvana po njezinom autoru, još je jedan primjer nebalansiranih podataka na kojima se provelo testiranje. Radi se o 96724 primjeraka zlonamjernog i 41323 benignog softvera, pri čemu su svi opisani s 54 značajke dobivene statičkom analizom pakiranih zlonamjernih datoteka preuzetih s `virusshare.com` te pakiranih benignih datoteka preuzetih s *Windows 2007*, *Windows XP* i *Windows 7* direktorija.

Koristeći ove podatke, pčelinjim programiranjem pronađen je model točnosti 98.07 % na testnom skupu podataka. Izgled njegovog stabla može se vidjeti na Slici 4.14, dok je napredak *fitness* funkcije najboljih modela kroz iteracije tog pokretanja algoritma prikazan

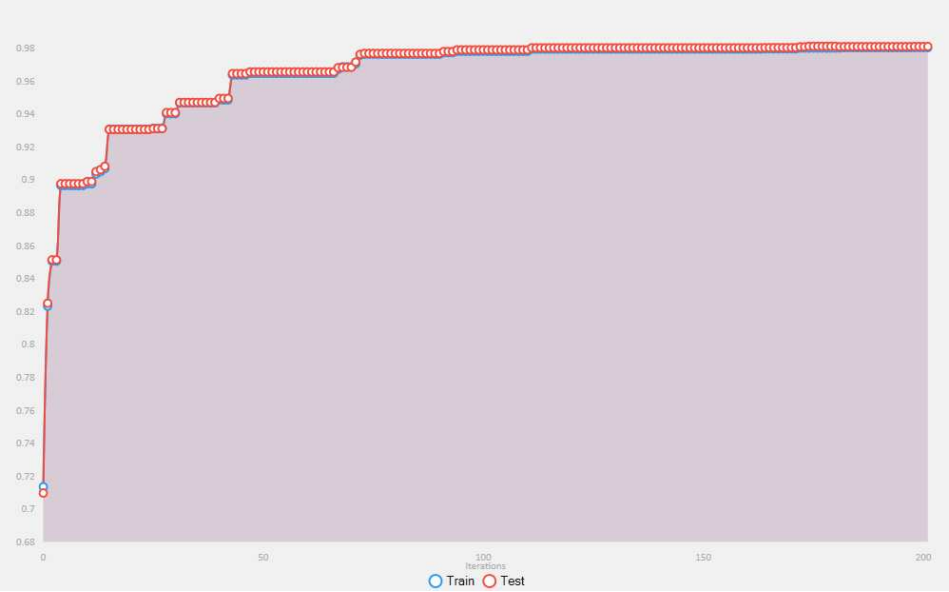
na Slici 4.15. U Tablici 4.4 su uspoređeni rezultati dobiveni pčelinjim programiranjem s rezultatima koje je dobio autor koristeći različite klasifikatore strojnog učenja.

Model	Točnost
Gaussov naivni Bayes	69.47
Stablo odluke	98.96
Nasumična šuma	**99.35**
AdaBoost	98.55
GradientBoosting	98.76
Pčelinje programiranje	**98.07**

Tablica 4.4: Usporedba točnosti različitih klasifikatora nad *Tek* podacima.



Slika 4.14: Izgled stabla globalno najboljeg modela za *Tek* podatke.



Slika 4.15: Napredak *fitness* funkcije najboljeg modela za *Tek* podatke kroz iteracije Algoritma 2.

4.2.4 *Android*

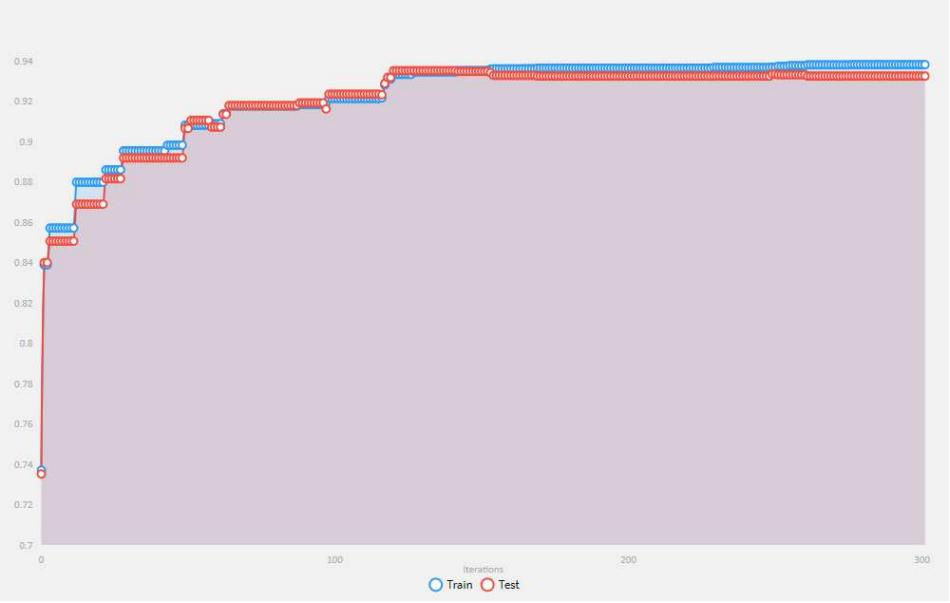
Kako bi se ispitale performanse pčelinjeg programiranja i na podacima koji nisu namijenjeni isključivo *Windows* uređajima, testirana je i baza podataka sačinjena od 215 značajki izvučenih statičkom analizom iz 3799 *Android* aplikacija [67]. Točnije, radi se o 1260 zlonamjernih aplikacija preuzetih iz *Android malgenome* [69] te 2539 benignih aplikacija.

Kako bi se smanjila složenost algoritma, broj značajki je prethodno reduciran na 55 korištenjem nasumičnih šuma.

Koristeći ove podatke, pčelinjim programiranjem pronađen je model točnosti 93.41 % na testnom skupu podataka. Napredak *fitness* funkcije najboljih modela kroz iteracije tog pokretanja algoritma prikazan je na Slici 4.15, dok su u Tablici 4.4 uspoređeni rezultati dobiveni pčelinjim programiranjem s rezultatima priloženim u [66].

Model	Preciznost (zlonamjernog softvera)	Odziv (zlonamjernog softvera)	Preciznost (benignog softvera)	Odziv (benignog softvera)	Težinska F1 mjera[33]
J48	0.972	0.964	0.984	0.9766	0.9766
REPTree	0.976	0.954	0.972	0.9786	0.973
Nasumično stablo - 100	0.975	0.978	0.987	0.985	0.9824
Nasumično stablo - 9	0.947	0.971	0.983	0.968	0.9692
Izglasani perceptron	0.969	0.95	0.971	0.982	0.9701
Većinsko glasanje	0.983	0.973	0.984	**0.99**	0.9837
Prosjek vjerojatnosti	0.983	0.973	0.984	**0.99**	0.9837
Maksimalna vjerojatnost	0.908	**0.996**	0.998	0.941	0.9617
MultiScheme	**0.984**	0.969	0.982	0.984	0.9784
Droid Fusion	0.981	0.984	**0.9991**	0.989	**0.9872**
Pčelinje programiranje	**0.9265**	**0.8933**	**0.9359**	**0.9565**	**0.9451**

Tablica 4.5: Usporedba kvalitete različitih klasifikatora nad *Android* podacima.



Slika 4.16: Napredak *fitness* funkcije najboljeg modela za *Android* podatke kroz iteracije Algoritma 2.

[33] Težinska F1-mjera dana je formulom $WFM = \frac{F_m \cdot N_m + F_b \cdot N_n}{N_m + N_b}$ gdje su F_m i F_b F1-mjere zlonamjernog, odnosno benignog softvera, a N_m i N_b njihov ukupan broj.

Zaključak

U ovom radu predstavljen je prethodno neupotrebljen način otkrivanja zlonamjernog softvera korištenjem meta-heurističkog pčelinjeg programiranja. Pristup je implementiran korištenjem Algoritama 2 i 5 te potom testiran na četiri različita skupa podataka. Iako se radi o novijem meta-heurističkom pristupu, rezultati testiranja pokazali su da korištenje pčelinjeg programiranja daje zadovoljavajuće rezultate u usporedbi s ostalim tehnikama korištenim u literaturi.

Poboljšanju rješenja, uz povećanje parametara veličine populacije i maksimalnog broja iteracija, zasigurno bi doprinijela i temeljitija upotreba unaprijeđenog pčelinjeg programiranja iz Algoritma 5. U ovom radu to nije bilo moguće ostvariti budući da se testiranje provodilo na većim skupovima podataka za što su potrebni kvalitetniji računalni resursi.

Bibliografija

[1] *Classification of malware pe headers*, https://github.com/urwithajit9/ClaMP, 2016, (pristupljeno 20. 10. 2020.)

[2] *Machine learning for malware detection*, https://github.com/Te-k/malware-classification, 2016, (pristupljeno 9. 11. 2020.)

[3] Monnappa K A, *Learning malware analysis*, paperback ed., Packt Publishing, 2018

[4] Sally Adee, *The global internet is disintegrating. what comes next?*, https://www.bbc.com/future/article/20190514-the-global-internet-is-disintegrating-what-comes-next, 2019, (pristupljeno 27. 9. 2020.)

[5] Suha Afaneh, Raed Zitar, and Alaa Hussein Al-Hamami, *Virus detection using clonal selection algorithm with genetic algorithm (vdc algorithm)*, Applied Soft Computing 13 (2013), 239–246

[6] J. Anderson, *Computer security technology planning study*, 1972

[7] Ömer Aslan and Refik Samet, *A comprehensive review on malware detection approaches*, IEEE Access 8 (2020), 1–1

[8] AV-TEST, *Malware statistics and trends report?*, https://www.av-test.org/en/statistics/malware/, (pristupljeno 27. 9. 2020.)

[9] John Aycock, , Advances in Information Security 22, Springer US, 2006

[10] Amin Azmoodeh, Ali Dehghantanha, Mauro Conti, and Kim-Kwang Raymond Choo, *Detecting crypto-ransomware in iot networks based on energy consumption footprint*, Journal of Ambient Intelligence and Humanized Computing (2017)

[11] Babak Bashari Rad, Maslin Masrom, and Suhaimi Ibrahim, *Camouflage in malware: From encryption to metamorphism*, International Journal of Computer Science And Network Security (IJCSNS) 12 (2012), 74–83

[12] Abhishek Bhattacharya and Radha Goswami, DMDAM: Data Mining Based Detection of Android Malware, 187–194, 11 2016, pp. 187–194

[13] CERT, *Analiza wannacry ransomwarea*

[14] ———, *Ddos napadi*

[15] ———, *Zlonamjerni softver*, (pristupljeno 27. 9. 2020.)

[16] Symantec Corporation, *Internet security threat report*, Tech. report, 2018

[17] ———, *Internet security threat report*, Tech. report, 2019

[18] Y. Dai, H. Li, Y. Qian, R. Yang, and M. Zheng, *Smash: A malware detection method based on multi-feature ensemble learning*, IEEE Access 7 (2019), 112588–112597

[19] K.-L Du and M.N.s Swamy, *Search and optimization by metaheuristics*, 08 2016

[20] Emmanuel Dufourq, *Data classification using genetic programming*, Master's thesis, School of Mathematics, Statistics, and Computer Science, University of KwaZulu-Natal, Pietermartizburg, 2 2015, Supervisor: Nelishia Pillay

[21] Melissa Eddy and Nicole Perlroth, *Cyber attack suspected in german woman's death*, https://www.nytimes.com/2020/09/18/world/europe/cyber-attack-germany-ransomeware-death.html, 2020, (pristupljeno 27. 9. 2020.)

[22] Edgar Galvan, Leonardo Trujillo, James Mcdermott, and Ahmed Kattan, Locality in Continuous Fitness-Valued Cases and Genetic Programming Difficulty, 41–56, 01 2013, pp. 41–56

[23] Daniel Gibert, Carles Mateu, and Jordi Planes, *The rise of machine learning for detection and classification of malware: Research developments, trends and challenges*, Journal of Network and Computer Applications 153 (2020)

[24] Beyza Gorkemli and Dervis Karaboga, *A quick semantic artificial bee colony programming (qsabcp) for symbolic regression*, Information Sciences 502 (2019), 346 – 362

[25] Aman Hardikar, *Malware 101 - viruses*, 2008

[26] IT Law Wiki, https://itlaw.wikia.org/wiki/The_IT_Law_Wiki, (pristupljeno 27. 9. 2020.)

[27] Gordon S. Novak Jr., *Automatic programming*, cs.utexas.edu/users/novak/autop.html, (pristupljeno 1. 10. 2020.)

[28] Dervis Karaboga, *An idea based on honey bee swarm for numerical optimization, technical report - tr06*, Technical Report, Erciyes University (2005)

[29] Dervis Karaboga, Celal Ozturk, Nurhan Karaboga, and Beyza Gorkemli, *Artificial bee colony programming for symbolic regression*, Information Sciences 209 (2012), 1 – 15

[30] Jinhyun Kim and Byung-Ro Moon, *New malware detection system using metric-based method and hybrid genetic algorithm*

[31] _____, *New malware detection system using metric-based method and hybrid genetic algorithm*

[32] Ondrej Kubovič, Peter Košinár, and Juraj Jánošík, *Can artificial intelligence power future malware?*, Tech. report, ESET, 2018

[33] Thi Le, Thi Chu, Quang Uy Nguyen, and Nguyen Hoai, *Malware detection using genetic programming*, 12 2014, pp. 1–6

[34] Mark Loman, *How ransomware attacks*, 2019

[35] Nikola Milošević, *History of malware*

[36] Francois Mouton and Arno de Coning, *Covid-19: Impact on the cyber security threat landscape*

[37] Jared Myers and Ed Murphy, *Technical analysis: Hackers leveraging covid-19 pandemic to launch phishing attacks, fake apps/maps, trojans, backdoors, cryptominers, botnets and ransomware*, https://www.carbonblack.com/blog/technical-analysis-hackers-leveraging-covid-19-pandemic-to-launch-phishing-attacks-trojans-backdoors-cryptominers-botnets-ransomware/, (pristupljeno 27. 9. 2020.)

[38] John Von Neumann and Arthur W. Burks, *Theory of self-reproducing automata*, University of Illinois Press, USA, 1966

[39] Angelo Oliveira, *Malware analysis datasets: Top-1000 pe imports*, 2019, (pristupljeno 5. 11. 2020.)

[40] Angelo Oliveira and Renato José Sassi, *Behavioral malware detection using deep graph convolutional neural networks*

[41] Younghee Park, D.s Reeves, and Mark Stamp, *Deriving common malware behavior through graph clustering*, Computers Security 39 (2013), 419–430

[42] Check point research, *Cyber attack trends analysis report*, Tech. report, 2019

[43] ———, *Cyber security report*, Tech. report, 2020

[44] Hrvatska regulatorna agencija za mrežne djelatnosti, *Što je 5g?*, https://www.hakom.hr/default.aspx?id=10321, (pristupljeno 27. 9. 2020.)

[45] Wasiur Rhmann, Babasaheb Bhimrao Ambedkar, and Gufran Ahmad Ansari, *Use of metaheuristic algorithms in malware detection*, 2017

[46] Royi Ronen, Marian Radu, Corina Feuerstein, Elad Yom-Tov, and Mansour Ahmadi, *Microsoft malware classification challenge*, CoRR abs/1802.10135 (2018)

[47] S. Abijah Roseline and S. Geetha, *An efficient malware detection system using hybrid feature selection methods*

[48] Wade Roush, *Inside the spyware scandal*, https://www.technologyreview.com/2006/05/01/229261/inside-the-spyware-scandal/, (pristupljeno 27. 9. 2020.)

[49] N. R. Sabar, X. Yi, and A. Song, *A bi-objective hyper-heuristic support vector machines for big data cyber-security*, IEEE Access 6 (2018), 10421–10431

[50] Shina Sheen, R. Anitha, and P. Sirisha, *Malware detection by pruning of parallel ensembles using harmony search*, Pattern Recognition Letters 34 (2013), 1679–1686

[51] Michael Sikorski and Andrew Honig, *Practical malware analysis: The hands-on guide to dissecting malicious software*, 1st ed., No Starch Press, USA, 2012

[52] SonicWall, *2020 sonicwall annual threat report*, Tech. report, 2020

[53] ———, *2020 sonicwall annual threat report mid-year update*, Tech. report, 2020

[54] Sophos, *Sophos 2019 threat report*, Tech. report, 2019

[55] ———, *Sophos 2020 threat report*, Tech. report, 2020

[56] Kenneth Sörensen and Fred W. Glover, *Encyclopedia of operations research and management science*, Springer US, Boston, MA, 2013

[57] Murugiah Souppaya and Karen Scarfone, *Nist special publication 800-83 revision 1, guide to malware incident prevention and handling for desktops and laptops*, Tech. report, 07 2013

[58] Alireza Souri and Rahil Hosseini, *A state-of-the-art survey of malware detection approaches using data mining techniques*, Human-centric Computing and Information Sciences 8 (2018), 1–22

[59] Peter Szor, *The art of computer virus research and defense*, Addison-Wesley Professional, 2005

[60] Rabia Tahir, *A study on malware and malware detection techniques*, International Journal of Education and Management Engineering 8 (2018), 20–30

[61] Ash Turner, *How many smartphones are in the world?*, https://www.bankmycell.com/blog/how-many-phones-are-in-the-world, 2020, (pristupljeno 27. 9. 2020.)

[62] John von Neumann, *The general and logical theory of automata*

[63] Liang Xiao, Yanda Li, Xueli Huang, and Xiaojiang Du, *Cloud-based malware detection game for mobile devices with offloading*, IEEE Transactions on Mobile Computing PP (2017), 1–1

[64] Yanfang Ye, Lingwei Chen, Shifu Hou, William Hardy, and Xin Li, *Deepam: a heterogeneous deep learning framework for intelligent malware detection*, Knowledge and Information Systems 54 (2017), 1–21

[65] Yanfang Ye, Dingding Wang, Tao Li, Dongyi Ye, and Qingshan Jiang, *An intelligent pe-malware detection system based on association mining*, Journal in Computer Virology 4 (2008), 323–334

[66] S. Y. Yerima and S. Sezer, *Droidfusion: A novel multilevel classifier fusion approach for android malware detection*, IEEE Transactions on Cybernetics 49 (2019), no. 2, 453–466

[67] Suleiman Yerima, *Drebin dataset*, https://figshare.com/articles/Android_malware_dataset_for_machine_learning_1/5854590/1, 2018, (pristupljeno 5. 11. 2020.)

[68] Pengtao Zhang, W. Wang, and Ying Tan, *A malware detection model based on a negative selection algorithm with penalty factor*, SCIENCE CHINA Information Sciences 53 (2010), 2461–2471

[69] Yajin Zhou and Xuxian Jiang, *Android malware genome project*, http://www.malgenomeproject.org/, 2012, (pristupljeno 12. 11. 2020.)

Sažetak

U ovom radu predstavljen je problem otkrivanja zlonamjernog softvera. Kako bi problem bio razumljiviji, obrađena je i tema definicije zlonamjernog softvera, njegovih kategorija, ali i njegova analiza te tehnike otkrivanja. Po prvi puta, za opisani problem, koristilo se meta-heuristično pčelinje programiranje čija implementacija je opisana u Poglavlju 3. Algoritam je testiran na četiri različita skupa podataka čiji rezultati su također prezentirani u ovome radu. Konačno, pokazalo se da pčelinje programiranje daje zadovoljavajuće rezultate pri rješavanju ovoga problema, ali ima i potencijala za njegov napredak.

Summary

This thesis presents the malware detection problem. In order to make the problem more understandable, the definition of malware, its categories, but also its analysis and detection techniques are also covered. For the first time, a metaheuristic artificial bee colony programming was used for the described problem. The implementation is presented in Chapter 3. The algorithm was tested on four different data sets, the results of which are also presented in this thesis. Finally, artificial bee colony programming has provided satisfactory results in solving this problem, but there is also potential for its progress.

Životopis

Rođena sam 19. siječnja 1997. godine u Novoj Gradiški, gdje sam pohađala osnovnu i srednju školu. Godine 2015. preselila sam se u Zagreb kako bih studirala na preddiplomskom sveučilišnom studiju Matematika na Matematičkom odsjeku Prirodoslovno-matematičkog fakulteta Sveučilišta u Zagrebu. Preddiplomski studij završila sam 2018. godine. Iste godine upisala sam diplomski sveučilišni studij Računarstvo i matematika.

Za vrijeme studiranja radila sam nekoliko studenskih poslova, od kojih je zadnji i trenutni u Ericsson-u s pozicijom studentskog konzultanta.