Sveučilište u Zagrebu
Fakultet elektrotehnike i računarstva

Diplomski rad br. 1638

Tehnike detekcije i obrane od zlonamjernih programa

Janko Cvitaš

Zagreb, prosinac 2006

Sažetak

Ovaj rad se bavi obranom od zlonamjernih programa. Navedene su vrste zlonamjernih programa i njihove općenite karakteristike. Osim toga, opisani su principi rada zlonamjernih programa, metode širenja i učinci koje mogu imati na inficirani sustav. Središte razmatranja su računalni virusi u kompiliranom obliku, posebice na *Win32* operacijskim sustavima, uz dodatan osvrt na računalne crve. Kao odgovor na maliciozne programe, razvijene su raznolike tehnike prepoznavanja koje omogućavaju osiguravanje sigurnog rada sustava. To uključuje više metoda pretraživanja pomoću uzoraka, heurističke i algoritamske pristupe, provjeru integriteta, sprječavanje infekcije usađivanjem, te emulaciju i zaustavljanje malicioznog ponašanja. U praktičnom dijelu ovog rada može se vidjeti implementacija jednostavnog primjera pretraživača, *PEScanner*, koji koristi *Boyer-Moore* algoritam za pretraživanje pomoću uzoraka, te strukturu *Portable Executable* formata izvršnih datoteka kao osnovu za heurističko pretraživanje. Korištenjem *Win32 API* funkcija ostvareno je i pretraživanje aktivnih procesa u memoriji, u korisničkom načinu rada.

Abstract

This work deals with defense against malicious software. Types of malicious software and their general characteristics are given in the text. In addition, the principles on which malicious software works are described, methods used for spreading and the effects it has on infected systems. The center of consideration are compiled computer viruses, especially on *Win32* operating systems, with an additional view on computer worms. As a response to malicious software, various techniques have been developed which enable secure system functioning. This involves several pattern-scanning methods, heuristic and algorithmic approaches, integrity checking, prevention of infection by innoculation, emulation and behavior blocking. As a practical part of this work, an implementation of a simple scanner example is given, *PEScanner*, which uses *Boyer-Moore* algorithm for pattern-scanning, and the structure of *Portable Executable* file format as a basis for heuristic scanning. *Win32 API* functions are used for the implementation of user-mode memory scanning of active processes.

Sadržaj:

1. Uvod ... 1
2. Zlonamjerni programi ... 1
   2.1 Uvod ... 1
   2.2 Formalni modeli ... 2
      2.2.1 John von Neumann: samoreplicirajući automat ... 2
      2.2.2 Edward Friedkin: reproduktivne strukture ... 3
      2.2.3 John Corton Conway: Igra života (Game of Life) ... 4
   2.3 Formalne i neformalne definicije ... 5
   2.4 Opće kategorije zlonamjernih programa ... 8
      2.4.1 Virusi ... 8
      2.4.2 Crvi ... 9
      2.4.3 Trojanski konji ... 10
      2.4.4 Logičke bombe ... 10
      2.4.5 Klice ... 11
      2.4.6 Izrabljivanje ranjivosti ... 11
      2.4.7 *Downloaderi* ... 11
      2.4.8 *Dialeri* ... 11
      2.4.9 *Dropperi* ... 12
      2.4.10 Generatori virusa ... 12
      2.4.11 *Keyloggeri* ... 12
      2.4.12 *Rootkits* ... 12
   2.5 Dodjeljivanje naziva malicioznim programima ... 13
   2.6 Zlonamjerni programi u različitim okruženjima ... 15
      2.6.1 Proučavanje okruženja rada *malware-a* ... 15
      2.6.2 Ovisnost o arhitekturi računala ... 15
      2.6.3 Ovisnost o centralnoj procesorskoj jedinici (*CPU*) ... 16
      2.6.4 Ovisnost o operacijskom sustavu ... 16
      2.6.5 Ovisnost o datotečnom sustavu ... 17
      2.6.6 Ovisnost o formatu datoteka domaćina ... 17
      2.6.7 Ovisnost o interpreterima ... 18
      2.6.8 Ovisnost o ranjivosti sustava ... 18
      2.6.9 Ovisnost o formatu arhiviranja ... 18
      2.6.10 Ostale ovisnosti ... 19
   2.7 Klasifikacija malware-a prema teretu ... 19
      2.7.1 Virusi bez tereta ... 19
      2.7.2 Virusi sa slučajno destruktivnim teretom ... 19
      2.7.3 Virusi sa nedestruktivnim teretom ... 19
      2.7.4 Virusi sa nisko-destruktivnim teretom ... 20
      2.7.5 Virusi sa visoko-destruktivnim teretom ... 20
      2.7.6 Napadi na dostupnost usluge (DoS) ... 21
      2.7.7 Krađa podataka ... 21
3. Tehnike napada *malware-a* ... 23
   3.1 Boot virusi ... 23

3.2 Infekcija datoteka ... 26
3.2.1 Virusi koji prepisuju postojeće datoteke ... 26
3.2.2 Virusi koji svoj kod dodaju na kraj inficiranih ... 27
3.2.3 Virusi koji svoj kod dodaju na početak inficiranih datoteka ... 28
3.2.4 Ameboidni virusi ... 29
3.2.5 Virusi koji popunjavaju „šupljine" u datotekama ... 30
3.2.6 Sažimajući virusi ... 31
3.2.7 Tehnika komadanja tijela virusa ... 31
3.3 WIN32 virusi ... 32
3.3.1 Win32 ... 32
3.3.2 PE format izvršnih datoteka ... 33
3.3.3 Infekcija zaglavlja ... 38
3.3.4 *Prepending* PE virusi ... 38
3.3.5 *Appending* PE virusi ... 38
3.3.6 Infekcija *kernel32.dll* datoteke. ... 38
3.3.7 Virusi pratioci ... 39
3.3.8 *Fractionated cavity* virusi na *Win32* ... 39
3.3.9 Napad na *lfanew* polje u zaglavlju ... 40
3.3.10 Virusi koji napadaju *VxD upravljačke programe* ... 40
3.4 In-Memory tehnike napada ... 41
3.4.1 Direktno aktivni virusi ... 41
3.4.2 Virusi rezidentni u memoriji na *MS DOS*-u ... 41
3.4.3 *Swapping* virusi ... 42
3.4.4 Virusi u korisničkom načinu rada (*Win32*) ... 42
3.4.5 Virusi u jezgrenom načinu rada (Windows NT/2000/XP) ... 43
3.5 Tehnike samoobrane zlonamjernih programa ... 44
3.5.1 Tehnika prikrivanja pomoću skoka ... 44
3.5.2 Prikrivanje ulazne točke virusa ... 45
3.5.3 Nevidljiivi virusi ... 46
3.5.4 Kriptirani virusi ... 47
3.5.5 Oligomorfni virusi ... 48
3.5.6 Polimorfni virusi ... 48
3.5.7 Metamorfni virusi ... 48
3.5.8 Retrovirusi ... 50
3.6 Računalni crvi ... 51
3.6.1 Općenita svojstva računalnih crva ... 51
3.6.2 Tragač mete ... 51
3.6.3 Prenositelj infekcije i izvršavanje koda ... 52
3.6.4 Ostale komponente računalnih crva ... 53
4. Tehnike obrane od malware-a ... 55
4.1 Pretraživači prve generacije ... 55
4.1.1 Pretraživanje nizova ... 55
4.1.2 Zamjenski znakovi ... 55
4.1.3 Ubrzavanje pretraživanja ... 56
4.2 Pretraživači druge generacije ... 56
4.2.1 Pametno pretraživanje ... 56
4.2.2 Detekcija kostura programa ... 57

4.2.3 Približno točna identifikacija ... 57
4.2.4 Točna identifikacija (*exact identification*) ... 57
4.3 Algoritamske metode pretraživanja ... 57
4.3.1 Značenje algoritamskih metoda ... 57
4.3.2 Detekcija dekriptora ... 58
4.4 Emulacija koda ... 59
4.5 Heurističke metode ... 60
4.6 Sprječavanje infekcije usađivanjem (cijepljenje) ... 62
4.7 Provjera integriteta ... 62
4.8 Zaustavljanje malicioznog ponašanja ... 63
5. Praktični rad ... 65
5.1 Uvod ... 65
5.2 Funkcionalnost programa ... 65
5.2.1 Izgled programa ... 65
5.2.2 Ispis zaglavlja ... 66
5.2.3 Pretraživanje datoteka ... 69
5.2.4 Pretraživanje memorije ... 73
5.2.5 Heurističko otkrivanje ... 74
5.3 Implementacija ... 75
5.3.1 Boyer-Moore algoritam ... 75
5.3.2 Pretraživanje sa zamjenskim (*wildcard*) znakovima ... 79
5.3.3 Pretraživanje memorije u korisničkom načinu rada ... 80
5.3.4 Heuristički pokazatelji ... 84
5.4 Testiranje programa ... 87
5.4.1 Testiranje pretraživanja pomoću potpisa ... 87
5.4.2 Testiranje heurističkih pokazatelja ... 89
5.4.3 Brzina izvođenja ... 91
5.5 Razmatranje o nadogradnji ... 92
5.5.1 Općenita razmatranja ... 92
5.5.2 Ubrzavanje pretraživanja pomoću *hash* funkcija ... 93
6. Zaključak ... 94
7. Literatura ... 95
8. Dodaci ... 97

1. Uvod

U ovom radu opisane su vrste malicioznih programa, te neki od načina na koji se oni šire i djeluju. Iz poznavanja tih metoda razvijaju se tehnike obrane od malicioznih programa koje obuhvaćaju raznolike pristupe od kojih su neki također opisani u ovom radu. Raznolikost malicioznih programa čini obuhvaćanje svih aspekata zaštite vrlo teškim zadatkom. Zbog toga su opisani općeniti pristupi problemu, koji mogu na različite načine biti implementirani u stvarnim programskim alatima za obranu od zlonamjernih programa.

Kao praktični dio rada implementiran je jednostavan pretraživač koji može pronaći neke vrste zlonamjernih programa i čija se izvedba temelji na nekim od pristupa opisanim u ovom radu. Iz opisa praktičnog dijela mogu se uočiti prednosti i mane jednog od mnogih pristupa ovom problemu.

2. Zlonamjerni programi

2.1 Uvod

U današnje doba gotovo da nema grane ljudskog djelovanja koja ne uključuje neku razinu korištenja računala. Korisnici su stručnjaci, tvrtke i pojedinci. Internetom se razmjenjuju jako velike količine podataka. U takvom okruženju rijetki su sustavi koji se ne susretnu sa nekim oblikom malicioznog softvera.

Razlozi stvaranja takvog softvera mogu biti raznoliki, ali u današnje doba nisu više toliko ni bitni. Činjenica je da sa razvojem tehnologije i otkrivanjem novih znanja rastu i mogućnosti autora malicioznog softvera. Moralne i etičke rasprave o zlonamjernom softveru prepuštaju se nekim drugim strukama, no na računarskim znanostima ostaje zadatak zaštiti korisnike od takvog softvera.

Kada se govori o malicioznom softveru i borbi protiv njega, ne može se govoriti o samo jednom izoliranom području računarstva. Obje strane se koriste cijelom paletom tehnika kako bi postigle svoj cilj. Najveći problem u borbi protiv takvog softvera je što su obično autori malicioznog softvera korak ispred znanstvenika koji se bave obranom od takvih programa. Često maliciozan program mora nastati kako bi nastala uspješna obrambena metoda protiv njega. Naravno sve više se koriste tehnike koje pokušavaju predviditi slijedeće poteze autora malicioznog softvera.

Kako bi bilo moguće stvoriti uspješnu obranu, potrebno je proučiti metode napada, ciljane objekte, mogućnosti sprječavanja djelovanja i mogućnosti oporavka nakon djelovanja takvog softvera.

2.2 Formalni modeli

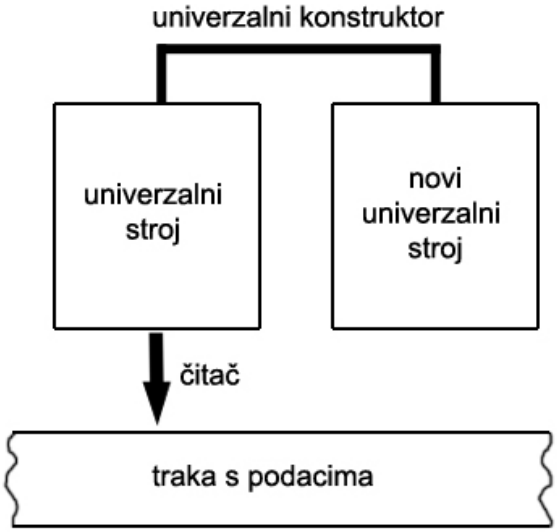
2.2.1 John von Neumann: samoreplicirajući automat

Iz potrebe da se opišu prirodne pojave i svijet oko nas ljudi se koriste modelima koji te pojave opisuju. John von Neumann je bio poznati matematičar i jedan od najbitnijih računalnih arhitekata u povijesti. Većina modernih računala današnjice temelje se na idejama koje je postavio von Neumann. Njegovi strojevi su uveli ideju memorije za pohranjivanje podataka te korištenje binarnih umjesto, dotada korištenih, analognih vrijednosti.

Tradicionalni von Neumannov stroj nije razlikovao podatke od instrukcija. Razlikovanje se odvijalo prema modu u kojem je stroj radio, te je o tome ovisilo hoće li se vrijednost prepoznati kao naredba ili podatak. U novijim su računalnim sustavima funkcije koje poboljšavaju razdvajanje instrukcija od podataka smatrane osnovom sigurnosti.

John von Neumann je prvi modelirao prirodnu samoreprodukciju koristeći ideju o samoreplicirajućem automatu 1948. godine, davno prije pojave prvih malicioznih samoreplicirajućih programa u računalnom svijetu. Osnova za takav model bio je univerzalni stroj (Turingov stroj).

Dijelovi sustava potrebnih za ovaj model:
- Univerzalni stroj
- Univerzalni konstruktor
- Traka sa podacima



Slika 2.1 Samoreplicirajući stroj

Koristeći univerzalni konstruktor Turingov stroj koristi elemente na traci i pomoću njih je sposoban izgraditi kopiju identičnu samom sebi. Pri tome stroj nije svjestan i ne razumije postupak nego prati upute koje su zapisane na traci. Pomičući „glavu za čitanje” po traci stroj traži idući odgovarajući dio. Kada pronađe zadani dio novog stroja, spaja ga sa prethodnima prema uputama također zapisanima na traci. Ukoliko svi potrebni dijelovi postoje stroj će biti u stanju izgraditi kopiju sebe. Kada je kopija izgrađena, novi stroj se pokreće i proces počinje ponovo.

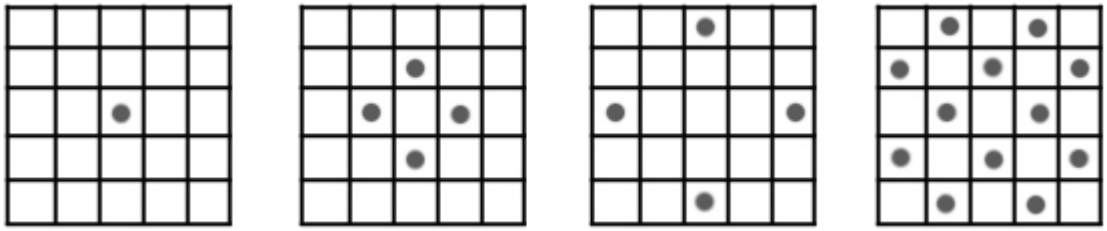
Daljnji korak razvoju modela samoreplicirajuće strukture von Neumannu je predložio Stanislaw Ulam, zagovarajući korištenje stanja ćelija umjesto dijelova stroja. Ćelije se upravljaju pomoću koda tj. instrukcija i poznate su pod nazivom *automat*. Nizovi ćelija nazivaju se *cellular automata* (automat sa ćelijama) arhitektura računala.

Novi von Neumannov model koristio se navedenim principom. Upotrebljene su ćelije koje su mogle postići 29 različitih stanja. Da bi stvorio model samoreplicirajuće strukture upotrebljeno je 200 000 ćelija. Sada je postojao matematički dokaz koji je potvrđivao mogućnost samoreplicirajućih struktura koje bi nežive elemente spajale u vlastite kopije, sa jednakim mogućnostima.

Arthur Burks je dovršio von Neumannov rad, a daljnji napredak je postigao E.F. Codd 1968. godine uspjevši pojednostaviti von Neumanov model. Novi model koristio je ćelije sa 8 stanja. Takvo pojednostavljenje je osnova za „samoreplicirajuće petlje” koje smanjuju opširnost univerzalnog stroja i usmjeravaju se na potrebe replikacije [2].

2.2.2 Edward Friedkin: reproduktivne strukture

Jedan od ljudi koji su pokušali pojednostaviti von Neumannov model bio je i Edward Friedkin. 1961 Friedkin je upotrijebio model opisan specijaliziranim automatom sa ćelijama u kojem su se sve strukture mogle reproducirati koristeći jednostavne uzorke na mreži.



Slika 2.2 Prve četiri generacije Friedkinovih struktura

Model se zasniva na slijedećim pravilima:

- U tablici koristimo jednu vrstu znački (*tokens*)
- Na svakoj poziciji imamo samo jednu ili nijednu značku

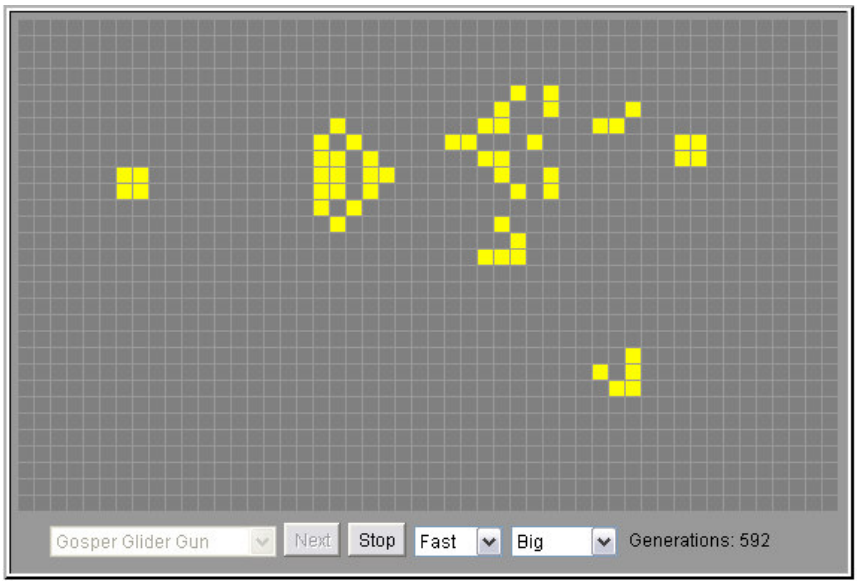
- Generacije znački prate jedna drugu u konačnom vremenskom okviru
- Okruženje svake značke odlučuje o tome hoćemo li imati novu značku u slijedećoj generaciji
- Okruženje označava polja iznad ispod lijevo i desno od značke (5-ćelijsko okruženje kakvo je koristio von Neumann)
- Polje će biti prazno u idućoj generaciji ukoliko u okruženju ima paran broj znački
- Polje će biti puno u idućoj generaciji ukoliko u okruženju ima neparan broj znački
- Moguće je promijeniti broj stanja

Koristeći ova pravila sve strukture se mogu reproducirati. [2]

2.2.3 John Corton Conway: Igra života (Game of Life)

1970. godine, britanski matematičar John Corton Conway stvorio je jedan od najčešće korištenih primjera automata sa ćelijama. Proučavanjem interakcija jednostavnih elemenata pod zajedničkim pravilima, otkrio je mogućnost stvaranja zanimljivih struktura. Nastala je igra *Life* koja zapravo ne zahtjeva ljudsku interakciju, nego samo početno zadano stanje. Model se sastoji od dvodimenzionalne mreže u kojoj je svako polje puno ili prazno (živo ili neživo), a promatra se okruženje od 8 polja (horizontalni, vertikalni i dijagonalni susjedi). Promjena stanja polja izvršava se prema slijedećim jednostavnim pravilima:

- Ukoliko je polje okruženo s manje od dva susjeda, polje umire
- Ukoliko je polje okruženo s više od tri susjeda, polje umire
- Ukoliko je polje okruženo s dva ili tri susjeda, ostaje nepromijenjeno
- Ukoliko je polje okruženo s točno tri susjeda, polje oživljava



Slika 2.3 „Igra života“ [10]

Posebno zanimljivo početno stanje je takozvana *Glider Gun* struktura. Nakon nekoliko početnih generacija dobiva se dojam da dvije strukture „pucaju” jedna na drugu pri čemu nastaju *glider-i* (klizač, jedrilica) koji se stvaraju i odlaze od osnovne strukture. Ovaj proces se nastavlja zauvijek stvarajući stalno nove *glider-e*. [2] [10]

2.3 Formalne i neformalne definicije

Matematičar Frederick Cohen smatra se „ocem” računalnih virusa. On je 1984, prvi uveo naziv *virus* na preporuku svog mentora, profesora Leonarda Adlemana koji je ime izabrao iz romana znanstvene fantastike.

Cohen je stvorio formalni model računalnog virusa. Za to se koristio Turingovim strojem. Njegov model vrlo je sličan von Neumannovom modelu samoreplicirajućeg automata sa ćelijama. Model nije od neke praktične pomoći u smislu proučavanja i obrane od stvarnih virusa, nego služi za pružanje formalne definicije i davanja teoretske osnove za ovaj rastući problem.

Cohen je dao i neformalnu definiciju računalnog virusa, koja se ne može shvaćati u najdoslovnijem smislu. Današnji virusi koriste razne taktike i metode napada, koje je teško obuhvatiti. Cohenova definicija glasi:

„Virus je program sposoban inficirati druge programe tako da ih modificira na način da sadrže eventualno evoluiranu kopiju tog virusa.”

Ova definicija daje grubu sliku što računalni virus predstavlja, no ne može se primijeniti doslovno. Razlog tome može se naći u jednostavnom primjeru *companion* (pratioc) virusa koji ne modificiraju kod datoteka domaćina. Umjesto toga oni iskorištavaju karakteristike okruženja u kojem napadnuti program radi, npr. operacijskog sustava, na način da se postavljaju u putanju izvršavanja ispred legitimnih programa. Striktno držanje za Cohenovu definiciju predstavljalo bi problem za antivirusne alate koji se oslanjaju na sprječavanje malicioznog ponašanja (*behavior-blocking*). *Companion* virusi ne mijenjaju strukturu datoteka domaćina i ne bi bili prepoznati kao prijetnja.

Cohen se u istraživanju obrane od virusa oslanjao na provjeru integriteta (*integrity-checking*) programa na računalu. Ta metoda se oslanja na bazu podataka koja sadržava podatke o „zdravom” stanju svakog programa koje se zabilježi u nekom inicijalnom trenutku koji smatramo sigurnim. Takva metoda također nije otporna na *companion* viruse s obzirom da oni ne mijenjaju integritet datoteka. Zbog toga se svaki program koji se pojavi na računalu, a nije u bazi podataka mora provjeriti nekom drugom metodom.

Osim lažnih negativnih rezultata, striktno držanje za Cohenovu definiciju može dovesti i do lažno pozitivnih rezultata. Postoji više primjera, a jedan od najočitijih su kompilatori (*compiler*) koji su predviđeni da mijenjaju kod drugih programa. Osim toga

programi za upravljanje datotekama (npr. *Norton Commander*) mogu imati ponašanje koje se kosi sa navedenom definicijom. Takvi program mogu kopirati sami sebe u druge direktorije, te moguće prepisati stariju verziju tog programa što se podudara sa definicijom. Takve greške nisu opasne i mogu se lako ukloniti, no mogu smetati korisniku.

Uzevši u obzir da nije bitno kako se virus kopira, i nije nužno da postoji zaražena datoteka domaćin, može se postaviti nešto općenitija definicija. No takva definicija i dalje obuhvaća i neke druge programe osim virusa i ne može se smatrati apsolutnim pokazateljem za prepoznavanje virusa. Općenitija definicija glasi:

„Računalni virus je program koji rekurzivno i eksplicitno kopira eventualno evoluirane kopije sebe samog."

Računalni virusi su samoupravljani programi koji se bez znanja korisnika kopiraju i šire na nova odredišta. Bitna je samo sposobnost da sami izvrše to kopiranje bez pomoći korisnika. Programi koji zahtjevaju korisnikovu akciju (kopiranje, umetanje u memoriju, promjenu koda datoteka…) ne mogu se smatrati računalnim virusima. Unatoč tome postoje programi koji pitaju korisnika želi li da se izvrše. Takvi programi jesu virusi zbog sposobnosti da sami sebe rekurzivno kopiraju [2].

Postoje i formalne definicije virusa, njihovog širenja i detekcije. Za to moramo promatrati skupove programa koji stvaraju druge programe. Za svaki par programa **p** i **q**, **p** u konačnom vremenu stvori **q** ako i samo ako **p** stvori **q** izravno ili nizom koraka. Virusni skup (*viral set*) je maksimalni skup programa **V** takav da za svaki par programa **p** i **q** u **V, p** stvori **q** u konačnom vremenu i **q** stvori **p** u konačnom vremenu. Maksimalni skup u ovom smislu znači takav da ne postoji program **r** izvan skupa **V** koji se može dodati u taj skup i zadovoljavati uvjete. Računalni virus se može smatrati virusnim skupom, a za program **p** se kaze da je instanca, tj. da je inficiran virusom **V** ako i samo ako je **p** član od virusnog skupa **V**. Za program koji je instanca nekog virusa kaže se da se širi svaki put kada stvori novu instancu iz tog virusnog skupa. Najjednostavniji virus je virusni skup koji sadrži točno jednu instancu virusa koji stvara samo samoga sebe. Veći skupovi predstavljaju polimorfne viruse koji se pojavljuju u više oblika.

Kako bismo definirali detekciju virusa, promatramo algoritam **A** koji detektira virus **V** ako i samo ako za svaki program **p** algoritam **A**(**p**) završi u konačnom vremenu, i daje pozitivan rezultat ako i samo ako je program **p** inficiran virusom **V**. Slično tome, algoritam **A** detektira skup virusa **S** ako i samo ako za svaki program **p** **A**(**p**) završi u konačnom vremenu, i daje pozitivan rezultat ako i samo ako je program **p** inficiran virusom **V** koji je član od **S**.

Cohen daje dokaze da ne postoji takav algoritam koji može otkriti skup svih mogućih računalnih virusa. Dokaz je sličan dokazu neprebrojivosti realnih brojeva. Za svaki algoritam za detekciju virusa **A** postoji program **p** koji glasi:

Ako **A**(**p**) *onda* završi, *inače* širi se;

Očito **A** ne vraća točan rezultat pozvan nad **p** budući da ako vrati pozitivnu vrijednost (kaže daje **p** inficiran) onda se **p** zaustavlja (što znači da nije inficiran). U suprotnom ako vrati bilo koju drugu vrijednost onda se **p** širi, što ukazuje na to da je inficiran. Ne postoji algoritam koji prepoznaje sve viruse bez pogrešnih rezultata, svaki program koji pokuša detektirati sve viruse će vratiti neke pogrešno pozitivne ili pogrešno negativne rezultate, ili uopće neće vratiti rezultat (greška u programu).

Postoji vrlo sličan dokaz koji pokazuje da postoje virusi za koje ne postoji algoritam koji ih otkriva bez pogrešnih rezultata. To znači da osim što ne možemo napisati program koji pronalazi sve poznate i nepoznate viruse bez pogrešnih rezultata, također postoje virusi za koje čak i ako imamo uzorak koji smo potpuno analizirali ne možemo napisati program koji ih detektira bez pogrešno pozitivnih rezultata.

Za virus se kaže da je polimorfan ukoliko njegov virusni skup ima više od jednog člana, tj. ako je kod tog virusa različit u različitim zaraženim objektima. Pretpostavimo virus koji je dovoljno polimorfan da za svaki algoritam **X** koji se može implementirati program **p** glasi:

```
Ako X(p) onda završi, inače širi se;
```

i za **p** vrijedi da je instanca virusa. Ne postoji algoritam **B** koji točno detektira virus, iz argumenta analognom prethodnom dokazu: za svaki algoritam **B** koji „tvrdi” da detektira ovaj virus postoji program **q**:

```
Ako B(q) onda završi, inače širi se;
```

za kojeg **B** ne vraća točan rezultat. Ako **B**(**q**) vrati pozitivnu vrijednost onda se **q** ne širi, što znači da nije instanca ovog ili bilo kojeg drugog virusa. Ako **B**(**q**) vrati negativnu vrijednost **q** se širi što znači da je **q** instanca virusa.

U stvarnosti postoji mogućnost postojanja dovoljno polimorfnog virusa takvog da za njega vrijedi prethodni dokaz. Uzmimo virus **W** sa instancom **r**:

```
Ako potprogram_jedan(r) onda završi, inače{
            Zamijeni kod od potprogram_jedan sa slučajno izabranim
programom;
            Proširi se;
            Završi;
      }

      Potprogram_jedan:
            Vrati false;
```

Slika 2.4 Polimorfni virus **W**

Za svaki algoritam **C** koji je kandidat za detekciju virusa **W** postoji program **s:**

```
      Ako potprogram_jedan(s) onda završi, inače{
            Zamijeni kod od potprogram_jedan sa slučajno izabranim
programom;
            Proširi se;
            Završi;
      }

      Potprogram_jedan:
            Vrati C(argument);
```

Slika 2.5 Program **s**

za koji **C** ne vraća točan rezultat. Ako **C**(**s**) vrati pozitivnu vrijednost onda se **s** ne širi, što znači da nije instanca od **W** ili bilo kojeg drugog virusa. Ako **C**(**s**) vrati negativnu vrijednost **s** se širi što znači da je **s** instanca virusa **W** [1] [3] [2].

2.4 Opće kategorije zlonamjernih programa

2.4.1 Virusi

Kao što je prije opisano, računalni virusi su maliciozni programi koji se rekurzivno repliciraju i šire se računalnim sustavima umetanjem kopija vlastitog koda u druge datoteke i ostale dijelove sustava. Ponašaju se slično virusima iz prirode koji inficiraju žive stanice. Kopiranje vlastitog malicioznog programskog koda u drugu datoteku naziva se „infekcija” (*infection*) a napadnuta datoteka „domaćin” (*host*).

Naziv „virus” često se koristi za sve vrste malware programa. Virusi koji su pušteni u opticaj tj. koji se neželjeno šire računalnim sustavima, nazivaju se virusi „u divljini” (*in the wild*), dok se virusi čuvani u kontroliranim laboratorijskim uvjetima radi istraživanja nazivaju „zoo-virusi”.

Program koji se smatra prvim *in the wild* virusom na mikroračunalima je „*Elk Cloner*”. Napisao ga je student Rich Skrenta 1982. godine, za računalo *Apple II*, misleći da neće raditi. *Elk Cloner* je ispisivao pjesmicu autora nakon svakih pedeset pristupa zaraženom disku kada je pritisnut reset. [2]

```
Elk Cloner:  The program with a personality

  It will get on all your disks
    It will infiltrate your chips
      Yes it's Cloner!

  It will stick to you like glue
    It will modify ram too
      Send in the Cloner!
```

Slika 2.6 Ispis *Elk Cloner-a*

2.4.2 Crvi

Računalni crvi (*worms*) su maliciozni samoreplicirajući programi koji se šire putem računalne mreže. Za razliku od virusa oni su najčešće zasebni programi i ne inficiraju datoteke domaćine, iako postoje i takvi slučajevi. Crvi koriste mrežu kako bi se raširili na udaljeni računalni sustav, najčešće koristeći propuste u dizajnu prijenosa podataka, bez znanja i interakcije korisnika sustava. Dok teret (*payload*) može imati razne učinke, glavna karakteristika je da crvi zagušuju mrežu i smanjuju propusnost, za razliku od virusa koji narušavaju integritet napadnutih (inficiranih) datoteka.

Naziv crv (*worm*) nastao je prema romanu znanstvene fantastike *The Shockwave Rider* autora Johna Brunnera. Prvi put je upotrebljen u članku istraživača Johna F. Shocha i Johna A. Huppa, koji su uočili sličnost sa programima na kojima su radili, 1982. godine.

Prvi primjer računalnog crva napravili su upravo dva navedena istraživača. Cilj je bio pronalazak neaktivnih procesora na mreži i ujednačavanja tereta, time poboljšavajući iskorištenost procesora (*CPU cycle use efficiency*). Ovaj program je bio ograničen tako da se nije mogao širiti izvan željenih granica i nije bio opasan.

Mailer i *mass-mailer* crvi su zasebna kategorija računalnih crva koji koriste slanje e-mail poruka sa malicioznim programom kada primatelj jedne takve poruke pokrene maliciozni program. Razlika između *mailer* i *mass-mailer* crva je frekvencija kojom šalju e-mail poruke sa uključenom kopijom samoga sebe. Primjeri su slanje e-mail poruka svim adresama pohranjenim u e-mail klijentima na računalu, slanje pri svakom korisnikovom slanju e-mail poruke itd.

Hobotnica (*octopus*) je sofisticiran oblik računalnog crva koji se sastoji od skupa programa raspoređenih na više računala. Tako se primjerice glava i krakovi *hobotnice* nalaze na udaljenim računalima i surađuju u izvršenju željene maliciozne funkcije. Ideja za ovakvu vrstu malicioznih programa potječe iz navedenog romana *The Shockwave Rider*. Iako nisu učestali, ovakvi crvi će se vjerojatno sve više pojavljivati u budućnosti, sa porastom kompleksnosti *malware*-a.

Zec (*rabbit*) je kategorija crva koji održava samo jednu kopiju sebe koja „skače" sa računala na računalo putem mreže. Druga upotreba ovog naziva je za programe koji se rekurzivno pokreću, ispunjavajući memoriju svojim kopijama. U sustavima na kojima rade i druge aplikacije koje nisu spremne na nedostatak memorije mogu uzrokovati znatnu štetu [2].

2.4.3 Trojanski konji

Jedan od najjednostavnijih oblika malicioznih programa su trojanski konji (*trojan horses*). To su programi koji sadržavaju korisnu funkcionalnost, ili se barem takvima čine, te pokušavaju na taj način privući korisnika da ih pokrenu i time omoguće izvršavanje malicioznog tereta (*payload*). U ostalim slučajevima, hakeri ostavljaju „trojanizirane" verzije originalnih alata na računalu kako bi im omogućio pristup i kontrolu nad zaraženim računalom. Postoje dvije vrste trojanskih konja, oni koji su stopostotni trojanski programi, i oni koji su modificirani postojeći programi.

Jedan od najpoznatijih trojanskih konja je *AIDS TROJAN DISK* koji je poslan na disketi preko 7000 istraživačkih organizacija. Jednom pokrenut, trojanac bi izmješao imena datoteka i ispunio sav prazan prostor na disku. Autor virusa je nudio obnavljanje podataka za novčanu naknadu, no uskoro je uhićen.

Stražnji ulaz (*backdoor*) je alat kojim se koriste zlonamjerni hakeri kako bi dobili pristup udaljenom računalu. Ti alati se najčešće nalaze unutar funkcionalnosti trojanskog konja. Pri korisnikovom pokretanju trojanskog konja, otvara se mrežni pristup (TCP/UDP) na kojem *backdoor* čeka instrukcije napadača. Drugi oblik *backdoor* pristupa je greška u dizajnu programa ili operacijskog sustava koji omogućava neželjeni pristup. Primjer je rana implementacija SMTP protokola koja je omogućavala izvršavanje naredbi.

Kradljivci lozinki (*password-stealing trojans*) su klasa trojanskih konja koji pamte korisnikove lozinke i šalju ih napadaču. Ova funkcionalnost je često u kombinaciji sa *keylogger* funkcionalnosti koja pamti pritiske na tipkovnicu, primjerice kod provjere pristupa računalu (*logon*) [2].

2.4.4 Logičke bombe

Logičke bombe (*logic bombs*) su dijelovi koda unešeni u inače legitiman softver, koje izvršavaju maliciozni teret (*payload*) pri ispunjenju određenih uvjeta. Može se raditi o programu koji sam sebe izbriše nakon što istekne rok za korištenje, ali može se raditi i o puno opasnijim teretima. Uvjeti koji iniciraju izvršavanje malicioznog koda mogu biti različiti, poput određenog datuma ili korištenja određenih riječi u radu sa alatom koji sadrži logičku bombu. Logičke bombe su slične „uskršnjim jajima" (*Easter eggs*) koja predstavljaju skrivene poruke autora softvera, ili dijela softvera koji žele „ostaviti svoj

trag" u softveru. Takve poruke često se teško otkrivaju, i nisu opasne. Logičke bombe se najčešće pojavljuju u velikim projektima čiji se kod rijetko ili nedovoljno provjerava.

Primjer logičke bombe je softver koji je stvorio zaposlenik *General Dynamics*-a 1992. godine koji je trebao izbrisati neke bitne podatke. Cilj je navodno bio da se navedeni zaposlenik ponovo zaposli kao visoko plaćeni savjetnik i riješi stvoreni problem. Njegov kolega naišao je na logičku bombu prije nego je pokrenuta i sprječio štetu. Krivac je uhićen i optužen [2] [19].

2.4.5 Klice

Klice (*germs*) su prva generacija virusa koji nastaju nakon kompiliranja koda virusa. Oni nisu u obliku u kojem se šire i nisu pridodani datoteci domaćinu. Njihovim pokretanjem nastaje druga generacija virusa čiji je kod (najčešće) umetnut u druge datoteke domaćine. Klice polimorfnih i kriptiranih virusa obično nisu kriptirane nego su u jasnom obliku.

2.4.6 Izrabljivanje ranjivosti

Izrabljivanje ranjivosti (*exploits*) je oznaka za programski kod specifičan za jednu ili skup ranjivosti određenog sustava. Cilj je iskorištavanje te ranjivosti i pokretanje programa na udaljenom računalu ili postizanje nekog drugog privilegiranog prava pristupa. Iako postoje maliciozni *exploit* kodovi, neki služe i za testiranje sigurnosti tj. mogućnosti probijanja sustava. Opasnost ovakvih programa ovisi o namjeri napadača.

2.4.7 *Downloaderi*

Downloaderi su oblik zlonamjernih programa koji pri pokretanju kopira druge maliciozne programe s interneta i pokreće ih, čime dolazi do infekcije računala na kojem je taj program pokrenut.

2.4.8 *Dialeri*

Dialeri su maliciozni programi koji su vrhunac doživili kada je većina ljudi koristila *dial-up* pristup internetu. Ovi programi bi koristili modem računala kako bi obavljali pozive na telefonske brojeve sa skupim tarifama i time zarađivali novac na štetu i bez znanja korisnika.

2.4.9 *Dropperi*

Dropperi su poput klica (*germs*) prva generacija virusa koja služi kao instalacijski program za programski kod virusa. Primjeri korištenja su virusi početnog sektora (*boot sector virus*) koji inficiraju sektore za punjenje (*boot sector*) diskova. Oni koriste *droppere* (koji su obični programi pohranjeni na disku) kako bi se prvi put upisali u *boot* sektor nekog diska i time pokrenuli svoje širenje.

Injectori su oblik *droppera* koji instaliraju kod virusa u memoriju. Nakon toga se virus širi uobičajenim načinima. Postoje i mrežni *injectori* koji koriste mrežu kako bi instalirali virus u memoriju udaljenog računala

2.4.10 Generatori virusa

Autori *malware-a* stvorili su velik broj programa koji automatski generiraju viruse sa željenim karakteristikama. Ti alati su jednostavni za korištenje što je uzrokovalo da i početnici mogu stvoriti maliciozan softver korištenjem jednostavnih menija takvih alata. Iako velik broj tako generiranih virusa ne postignu željenu destruktivnu namjeru, postoje i primjeri vrlo opasnih generiranih virusa poput *Anna Kournikova* virusa kojeg je stvorio nizozemski teenager Jan de Wit.

2.4.11 *Keyloggeri*

Keyloggeri su maliciozni programi koji zapisuju pritiske na tipkovnicu pokušavajući time pronaći osobne podatke i lozinke korisnika, te ih poslati napadaču. Ti podaci se mogu dalje koristiti za krađu, neovlašten pristup, itd.

2.4.12 *Rootkits*

Rootkits je općenit naziv za skup programa koji omogućava tzv. „root” pristup računalu tj. pristup sa najvišim ovlastima. Najčešće hakeri koriste *exploit-e* kako bi instalirali takve programe na napadnuto računalo, poput *trojaniziranih* verzija uobičajenih programa, što im kasnije omogućava neovlašten upad na računalo. Sofisticiranije verzije *rootkit* programa imaju i elemente koji rade u jezgrenom načinu rada što ih čini posebno opasnima s obzirom da mogu mijenjati ponašanje jezgre (kernela) operacijskog sustava i time se kamuflirati. Za razliku od toga *rootkit* programi koji rade u korisničkom načinu rada (*user-mode rootkits*) lakše se otkrivaju sustavima obrane koji rade u jezgrenom načinu rada (*kernel mode)* [20].

2.5 Dodjeljivanje naziva malicioznim programima

Organizacija CARO (*Computer Antivirus Researchers Organization*) izradila je shemu za pridavanje imena malicioznim programima. Ta specifikacija je nastala 1991. godine, ali s obzirom na česte pojave novog *malware-a* u novije vrijeme nije sasvim jednaka uobičajenoj praksi. Tvrke koje proizvode antivirusni softver uglavnom se pokušavaju držati predložene sheme, ali obično postoje razlike u imenima različitih proizvođača. Naime, količina od 500, 1000 ili više novih inačica *malware-a* koje proizvođači moraju pridodati svome softveru mjesečno čini zadatak praćenja zadavanja zajedničkog imena gotovo nemoguć [14].

Forma za imenovanje malicioznih programa prema organizaciji CARO izgleda ovako:

```
<malware_type>://<platform>/<family_name>.<group_name>.
  <infective_length>.<variant><devolution><modifiers>
```

To je najopširnija verzija za imenovanje, a slijedi opis dijelova te sheme:

`<family_name>` (ime porodice)

Ovo je osnovna komponenta imena *malware-a*. Pravila za zadavanje ovog dijela su:

- Ne koristiti imena tvrtki, *brandova*, ili živućih ljudi
- Ne koristiti postojeće ime familije *malware-a* ako osim ako virus pripada istoj porodici
- Ne koristiti uvredljiva imena
- Ne koristiti novo ime ako već postoji neko ime za tu porodicu
- Ne koristiti numerička imena
- Izbjegavati ime predloženo ili željeno ime
- Izbjegavati imenovanje prema datoteci koja najčešće sadržava taj *malware*
- Izbjegavati imena koja predstavljaju logički okidač za maliciozni teret (*payload*)
- Izbjegavati zemljopisna imena prema mjestu otkrića
- Ako postoji više prihvatljivih imena, izabrati ono koje koristi većina antivirusnih programa ili najdeskriptivnije

`<malware_type>://` (tip *malware-a*)

Ovaj dio imena označava radi li se o virusu, trojanskom konju, *dropperu* ili nekom drugom priznatom obliku *malware-a*. Priznate vrijednosti su ‘virus’, ‘trojan’, ‘dropper’, ‘intended’, ‘kit’ i ‘garbage’. Više proizvođača koristi širi skup od navedenog, i vjerojatno će se u budućnosti neke vrijednosti naći unutar standarda.

`<platform>/` (platforma)

Prefiks platforme označava minimalno okruženje koje je potrebno da bi *malware* radio. CARO daje listu standardnih platformi (dodatak 1.)

`<group_name>` (ime grupe)

Označava veću grupu porodica virusa koje su slične jedna drugoj, danas se rijetko koristi, a nekada se najčešće koristila za *DOS* viruse.

`<infective_length>` (infektivna duljina)

Infektivna duljina se koristi za razlikovanje parazitskih virusa unutar porodice ili grupe na temelju njihovih duljina u *byte-ovima.*

`<variant>` (varijanta)

Označava podvarijante unutar iste porodice sa istom infektivnom duljinom.

`[<devolution>]` (prijenos)

Devolution identifikator se obično koristi sa imenom podvarijante makro virusa. Neki makro virusi imaju zajedničku mogućnost (programske greške) da stvore podskup originalnog makro virusa putem ciklusa razmnožavanja.

`<modifiers>` (modifikatori)

Originalna namjera za polje modifikatora je bila identificiranje polimorfnog stroja računalnog virusa. No, danas se rijetko koristi u praksi. Sadrži i dodatne komponente:

```
[[:<locale_specifier>][#<packer>][@'m'|'mm'][!<vendor-
specific_comment>]]
```

`:<locale_specifier>` (specifikator lokacije)

Ovaj specifikator se najčešće koristi kod makro virusa koji ovise o posebnoj verziji jezika njihovog okruženja (poput francuskog *Word-a).*

`#<packer>`

Pokazuje je li *malware* pakiran sa posebnim *on-the-fly* ekstraktorom poput *UPX-a.*

@m ili @mm

Ovi simboli označavaju *mailer* ili *mass-mailer* viruse (crve). To je jedan od najčešće upotrebljavanijih modifikatora.

!<vendor-specific_comment>

Modifikator koji omogućava proizvođaču da doda željeni komentar kao sufiks imenu *malware-a*.

2.6 Zlonamjerni programi u različitim okruženjima

2.6.1 Proučavanje okruženja rada *malware-a*

Za proučavanje *malware-a* potrebno je imati razumijevanje o predviđenom okruženju za rad takvih programa. To se odnosi na razne specifične uvjete koji moraju biti zadovoljeni da bi svaki maliciozni program uspio inficirati određeni računalni sustav. Za svaki niz simbola (instrukcija) možemo definirati okruženje potrebno da takav program radi, no u praksi je potrebno pronaći stvarno okruženje kako bi se mogao stvoriti odgovarajući obrambeni sustav.

Automatizacija analize malicioznog koda postaje sve težim zadatkom zbog sve raznolikijih ovisnosti vezanih uz okruženje. Istraživaču *malware-a* potreban je velik trud kako bi našao uvjete u kojima se maliciozni kod može replicirati. Što je maliciozni program „lošiji" to je manje vjerojatno da istraživač pronađe uvjete u kojima takav program može biti opasan. O okruženju ovisi način analize koda pa je jasna važnost prepoznavanja ovisnoti o njemu [2].

2.6.2 Ovisnost o arhitekturi računala

Većina virusa se širi u binarnom (*binary*), tj. kompiliranom kodu. Primjerice *boot* virusi se šire tako da umetnu svoj kod u *boot* sekvencu računala. Neki od prvih virusa *Elk Cloner* i *Brain* bili su *boot* virusi, i iako su radili na sličnom principu, razlikovali su se s obzirom na arhitekturu računala (uključujući i ovisnost o CPU, korištenju memorije i slično). *Elk Cloner* bio je virus za *Apple*, a *Brain* za *IBM PC* računala. Prijelaz na stranu arhitekturu bio je nemoguć.

Iako je moguće stvoriti virus koji radi na više platformi, radi se o teškom zadatku. Potrebno je prepoznati arhitekturu, i ovisno o tome pokrenuti dio koda predviđen za nju. Dokaz da je to moguće pokazao je *PeElf* virus 2001. godine koji je prvi *cross-arhitecture* virus.

Ovu ovisnost (i ovisnost o operacijskom sustavu) autori *malware-a* zaobišli su drugim metodama. Koristi se pseudokod koji se prema potrebi prevodi u oblik prikladan za lokalno okruženje.

Postoje virusi koji se suzdržavaju od infekcije ovisno o određenom okruženju. Tako je primjerice virus *Cascade* iz 1987. godine odustajao od infekcije ako bi u *BIOS-u* našao oznaku *IBM copyright-a.* No, dio koda koji je time upravljao imao je grešku pa se virus širio na sve sustave.

Druge vrste virusa ovise o svojstvima *BIOS-a.* Kod *BIOS-a* koji se mogu nadograditi (*flashable, upgradeable*), postoji šansa za infekciju *BIOS-a* [2].

2.6.3 Ovisnost o centralnoj procesorskoj jedinici (*CPU*)

Ova ovisnost pogađa kompilirane računalne viruse. Izvorni programski kod se kompilira i pohranjuje u datoteke poput *EXE* formata. Tako kompilirani kod sadrži slijed instrukcija potrebnih za replikaciju virusa. Instrukcije su predstavljene brojevima koji imaju određeno značenje za svaki *CPU*. Zbog različitih tumačenja istih brojeva (*opcode*) virus koji se pokuša pokrenuti u stranom okruženju biti će krivo protumačen i program neće raditi. Tako je primjerice kod za *NOP* instrukciju na *Intel CPU* 0x90 dok je na *VAX CPU* kod za tu instrukciju 0x01.

Dodatna ovisnost o centralnoj procesorskoj jedinici može se uočiti kada novije verzije nekog procesora nisu sasvim kompatibilne sa starijim verzijama (*backward-compatible*). Tako novije verzije mogu podržavati drugačiji set instrukcija. Primjer je Intel 386 procesor na kojem postoji greška pri izvršavanju *CALL SP* instrukcije, koju su virusi koristili za predavanje kontrole kriptiranom dijelu svog koda. Osim grešaka (nije tako učestala pojava) neki novije verzije procesora promijene značenja nedokumentiranih funkcija, poput *POP CS* instrukcije koja je radila na *Intel 8086* procesoru, ali nije bila dokumentirana. Kasnije je proizvođač promijenio njeno značenje [2].

2.6.4 Ovisnost o operacijskom sustavu

Nekadašnji operacijski sustavi bili su predviđeni za rad na samo jednoj *CPU* arhitekturi. Tako je *MS DOS* radio isključivo na *Intel* procesorima. Prvi *Microsoft-ov* operacijski sustav koji je podržavao više arhitektura bio je *Windows NT*. Većina virusa ovisi o operacijskom sustavu. Tako primjerice virusi za 32-bitne *Windows* platforme inficiraju samo *PE(portable executable)* format datoteke, te neće moći raditi u *MS DOS* operacijskom sustavu. Obrnuto, neki *MS DOS* virusi koriste svojstva koja su nasljeđena u novije *Microsoft-ove* operacijske sustave te će raditi na njima. Neka svojstva su također uklonjena pa neki *MS DOS* virusi neće raditi na *Windows* operacijskim sustavima. Širenje virusa koji inficira isključivo *Windows* formate za izvršne datoteke na primjerice *Linux* operacijski sustav je nemoguće jer *Linux* koristi drugačiji format za izvršne datoteke.

Postoje tzv. *multipartite* virusi koji inficiraju više formata datoteka ili dijelova sustava, pa je za njih moguće da se prošire sa jednog operacijskog sustava na drugi.

Neki virusi ovise o verzijama pojedinog operacijskog sustava. Tako se mogu oslanjati na jezične postavke operacijskog sustava i time slučajno (ili namjerno) ciljati sustave određenog naroda. Takvi virusi zadaju probleme istraživačima kojima može biti vrlo teško pronaći verziju sustava na kojoj takav maliciozni program radi [2].

2.6.5 Ovisnost o datotečnom sustavu

Većina virusa ne ovisi o datotečnom sustavu (npr. FAT, NTFS), već koriste „*high-level*" sučelje operacijskog sustava kako bi koristili postojeće datoteke bez znanja o kojem se datotečnom sustavu radi. No neke vrste ovise izravno o datotečnom sustavu.

Klaster (*cluster*) virusi ovise o *FAT (File Allocation Table)* datotečnom sustavu. U tom sustavu, datoteke se na disk spremaju u klasterima, a tablica alokacija (*FAT*) se koristi kako bi se pronašla pojedina datoteka. Izravnim pristupom disku moguće je izmjeniti pokazivač u *FAT* tablici. Originalni pokazivač se pohranjuje u neiskorištenom prostoru u *FAT* strukturi dok se na njegovo mjesto postavlja pokazivač na program virusa. Pri pokretanju željenog programa, umjesto njega se iz *FAT* tablice učitava adresa virusa i on se pokreće. Kad je jednom pokrenut, virus može pokrenuti originalni program i time zamaskirati svoje djelovanje. Ovakvi virusi postoje u samo jednoj instanci na računalu koja se pokreće pri pokretanju svakog drugog programa. Ovakvi virusi se nazivaju „super brzim" infektorima.

Na NTFS (*NT File System*) datotečnom sustavu svaka datoteka može sadržavati višestruke tokove (*stream*). Osnovni tok predstavlja samu datoteku, dok se u istu datoteku može pohraniti dodatni tok. Takvi virusi mogu inficirati datoteku tako da njen kod prepišu svojim, ali da prije toga pohrane original u dodatni tok koji se kasnije poziva kako bi se normalno izvršio inficirani program. Neki virusi se oslanjaju na NTFS kompresiju i ne mogu raditi bez nje.

Iako nije učestala pojava, virusi mogu inficirati „*image"* formate CD-ova, poput *ISO 9660* standarda. Virusi inficiraju *ISO* format prije snimanja na medij, a mogu se jednostavno širiti korištenjem primjerice *AUTORUN.INF* datoteke koja se automatski izvršava na *Windows* operacijskom sustavu [2].

2.6.6 Ovisnost o formatu datoteka domaćina

COM virusi na *DOS* operacijskom sustavu inficiraju isključivo izvršne datoteke sa ekstenzijom .*COM*. Takve datoteke nemaju određenu strukturu pa su jednostavna meta za viruse.

EXE virusi na *DOS* operacijskom sustavu inficiraju *EXE* datoteke koje na početku sadržavaju jednostavno zaglavlje koje operacijski sustav koristi pri izvršavanju. Zaglavlje počinje slovima „*MZ*" prema inicijalima autora ovog formata. (neke verzije prihvaćaju i ZM). Neki virusi mijenjali su tu oznaku u „*ZM"* kako bi otežali posao antivirusnom softveru koji traži datoteke sa „*MZ*" oznakom i kako bi označili inficiranu datoteku i spriječili višestruku infekciju.

16-bitni *Windows-i* i *OS2* koriste *NE* (*New Executable*) format izvršnih datoteka. Njihova struktura je nešto složenija od navedene *EXE* strukture, i nakon početnog *EXE* zaglavlja sadrži dodatno zaglavlje koje se može prepoznati po „*NE*" oznaci. Neki virusi

iskorištavaju kompleksnost ovog formata da bi otežali dezinfekciju i popravak inficirane datoteke.

Windows operacijski sustavi iz NT porodice koriste *PE (portable* e*xecutable)* format izvršnih datoteka. One također sadrže *DOS EXE* zaglavlje nakon kojeg je *PE* zaglavlje. Infekcija *PE* datoteka je najzastupljenija metoda širenja današnjih virusa. Osim izvršnih datoteka, u istom formatu su i mnoge druge poput *DLL* (*dynamic link library)* datoteka. One sadržavaju funkcije koje ostali programi mogu pozivati (u svrhu modularnosti). Neki virusi mogu promjenom tablice izvezenih (*exported*) funkcija jednostavno inficirati tj. zakačiti se na sučelja *API(Application Programing Interface*) funkcija

Virusi koji rade u *Unix/Linux* okruženjima rade na sličnom principu. Izvršni fomrat na *Unixu* je *ELF* koji također sadrži zaglavlje, te je tijelo datoteke podijeljeno u logičke cjeline [2].

2.6.7 Ovisnost o interpreterima

Velik broj virusa ovisi o interpreterima ugrađenima u određeno okruženje. Neki programi imaju svojstvo programabilnosti, tj. korisničkog određivanja ponašanja. Jedan od takvih primjera je *Microsoft Office* paket koji pruža mogućnosti programiranja makro naredbi. Te naredbe tj. programi mogu imati i neželjene efekte za korisnika, te mogućnost širenja prema definiciji virusa.

Neki virusi se oslanjaju na interpreterske jezike čiji se kod ne kompilira već se izvodi korištenjem interpretatorskog modula. Primjeri su programski jezici poput *PHP-a, Python-a, PERL-a* i mnogih ostalih [2].

2.6.8 Ovisnost o ranjivosti sustava

Neki brzo-propagirajući crvi se oslanjaju isključivo na širenje u slučaju postojanja određene ranjivosti na sustavu. Popravljeni (*patched*) sustavi su imuni na takve napade, ali postoje i maliciozni programi koji traže jednu iz većeg skupa ranjivosti. Ukoliko jedna od ranjivosti postoji na sustavu, taj *malware* će ga uspješno inficirati [2].

2.6.9 Ovisnost o formatu arhiviranja

Neki virusi ne mogu raditi bez arhiviranih datoteka poput *ZIP, RAR, ARJ* formata. Neki virusi čak koriste arhivirane formate zaštićene lozinkom. Ta lozinka je dostupna korisniku primjerice u e-mail poruci, dok ju antivirusni softver ne može dohvatiti. Korisnik se može prevariti kako bi otpakirao takvu datoteku i pokrenuo maliciozni program (npr. *mass-mailer malware* koji šalje takvu arhiviranu kopiju novim žrtvama putem e-maila [2].

2.6.10 Ostale ovisnosti

Gotovo je nemoguće naći dio sustava, koji neki *malware* ne pokušava iskoristiti za svoje širenje ili izvršavanje. Osim navedenih potrebno je spomenuti i ovisnosti *malware-a* o: *JIT (Just In Time)* kompilaciji, izvornom kodu programa (*source code*), mrežnim protokolima, kompilatorima, *debuggerima,* itd. [2].

2.7 Klasifikacija malware-a prema teretu

2.7.1 Virusi bez tereta

Iako većina štete na računalima nije uzrokovana virusima, također postoje virusi koji ne sadrže teret osim mogućnosti replikacije (*no-payload viruses*). No unatoč tome niti jedan virus se ne može smatrati bezopasnim. Velik broj virusa sadrže poruke koje se nikada neće pojaviti običnom korisniku, već su predviđene samo za istraživače antivirusnih tvrtki. Neki virusi ne nose apsolutno nikakav teret. No i tehnika širenja može imati destruktivna svojstva, poput pisanja preko originalnog koda programa. Čak i virusi koji ne diraju originalni kod datoteka domaćina mogu prouzročiti štetu, primjerice gubitak podataka uslijed rušenja sustava uzrokovanog greškom u kodu virusa. Uspješan napad virusa bez destruktivnog tereta u sustave tvrtki koje se bave proizvodnjom softverskih ili hardverskih sustava može zaustaviti proizvodnju na duže vrijeme i uzrokovati veliku novčanu štetu [2].

2.7.2 Virusi sa slučajno destruktivnim teretom

Neki virusi poput *Stoned* boot [2] virusa uzrokuju gubitak podataka svojim procesom replikacije. *Stoned* čuva originalni boot sektor tako da ga pohranjuje na kraj *root direktorija.* Na taj način dolazi do gubitka nekih zapisa ako ima puno tih zapisa upisanih u *root direktorij*. Korisniku će se umjesto imena datoteka prikazati nečitljiv tekst. Datotetke je moguće vratiti, ali to zahtjeva malo stručniji angažman. Takvi virusi nazivaju se slučajno destruktivnima (*accidentally destructive viruses*) [2].

2.7.3 Virusi sa nedestruktivnim teretom

Gotovo polovica virusa pripada ovoj kategoriji. Autori *malware-a* su često politički motivirani i cilj im je ispisati svoje poruke korisnicima. Osim što smetaju korisnicima, takvi virusi ne uništavaju druge dijelove sustava, osim možda procesom replikacije (*no-payload*). Neki virusi koriste tekstualna sučelja za ispisivanje svojih poruka ili animacija. Neki koriste grafička sučelja i unaprijed dizajnirane animacije kako bi prikazali svoju prisutnost. Osim vizualnih tereta koriste se i ostali, poput puštanja zvukova kroz zvučni

sustav računala, ili korištenja fizičkih komponenti računala poput otvaranja i zatvaranja ladica CD uređaja.

Za demonstraciju može se spomenuti *W95/Haiku* virus koji se pri aktivaciji spajao na određenu IP adresu na Internetu, skidao slučajno izabranu *.wav* datoteku i puštao snimku haiku pjesme [2].

2.7.4 Virusi sa nisko-destruktivnim teretom

Primjer ovakvog virusa je *W95/HPS*. On se aktivira isključivo subotom, te tada horizontalno okrene sve slike u čistom *bitmap* formatu koje *Windowsi* dohvati. Tako obrnute slike označi unutar *bitmap* zaglavlja. Na taj način prepozna već napadnute slike i ne može se dogoditi da ih vrati u originalni oblik.

U ovu skupinu može se smjestiti i velik broj retro-virusa, koji napadaju antivirusne programe. Antivirusi sadrže niz znakova pomoću kojih prepoznaju i uništavaju viruse. Retro-virusi uništavaju samo takve datoteke, dok ostale ne diraju. Moguće je da retro-virus pošalje aktivnoj antivirusnoj aplikaciji poruku o „*Windows shutdown*", te će se aplikacija ugasiti „misleći" da mora prije gašenja operacijskog sustava.

Virus *WM/Wazzu.A.* napada tekstualne dokumente te zamijeni tri slučajno odabrane riječi sa riječi „*wazzu*". Radi se o makro virusu, koji se proširio toliko da je čak i *Microsoft* izdao nekoliko CD-ova sa datotekama koje su bile zaražene [2].

2.7.5 Virusi sa visoko-destruktivnim teretom

Postoje mnogi virusi kojima je namjera uništiti korisnikove podatke (*highly-destructive payload*). Neki od njih formatiraju diskove ili prepišu podatke na njemu. Jedan od prvih takvih zloglasnih virusa je *Michelangelo*, koji je brisao dijelove particije sa koje podizao sustav. Datoteke su mogle biti vraćene. Neki virusi brisali bi samo glavni zapis za učitavanje *(master boot record - MBR)* što bi uzrokovalo nemogućnost podizanja sustava. Mnoge tvrtke su naplaćivale obnovu podataka prema količini vraćenih podataka, što je bilo dosta skupa usluga zbog samo jednog sektora diska (*MBR*).

Virus *Hungarian Filler* osim što je brisao *FAT* sektore, ispunjavao bi ih uzorkom znakova „☺". Njegova aktivacijska rutina enkriptirana je unutar tijela virusa te zbog toga nikad nije dokumentirana.

Neki virusi poput onih iz *AntiPascal* porodice jednostavno brišu datoteke. Cilj su im bili programi u pisani u *Pascal-u*.

Poseban naziv *data diddlers* je skovan za skupinu virusa koji uništavaju podatke, ali na polagan način koji nije očit korisniku. *Dark_Avenger.1800.A* virus polako bi

zapisivao određen tekst na slučajno odabrane sektore na disku. Na taj način sustav bi polako zatajivao, ali virus bi se i dalje širio. Do trenutka kada bi korisnik shvatio da je njegov sustav inficiran, velik broj podataka bi bio uništen zapisanim tekstom. Virus *Ripper* djeluje tako da zamjenjuje dvije *WORD* vrijednosti u slučajno odabranom sektoru diska. Takav pristup u teoriji može čak dovesti do mutacije nekog drugog virusa. Ripper bi u tom slučaju zamijenio dvije instrukcije koda virusa, čime ne bi narušio kod (nije vrlo vjerojatno), ali bi mu promijenio strukturu i time onemogućio antivirusne programe da ga detektiraju.

Neki virusi koriste kriptiranje podataka kako bi uzrokovali štetu. Tako je moguće samo kriptirati podatke što čini teškim za korisnika da ih vrati. Osim toga kriptiranje podataka korisnika služi za ucjenu, autori takvog *malware-a* nude ključ za dekriptiranje podataka u zamjenu za novac (ili neku drugu vrijednost). Treći oblik korištenja enkripcije podataka je stvaranje određene simbioze između sustava i virusa čime je uvelike otežano njegovo uklanjanje. Takav virus kriptira podatke, no radi u memoriji, i omogućava korisniku korištenje takvih podataka pri svakom pristupu. No ukoliko se virus odstrani iz sustava, podaci ostaju kriptirani i korisnik više ne može do njih [2].

2.7.6 Napadi na dostupnost usluge (DoS)

Putem interneta moguće je izvršiti ciljani napad protiv određene tvrtke ili dostupnosti sustava (*DoS – Denial of Service*). Najčešće takve napade izvršavaju računalni crvi. Iako neki od njih ne napadaju isključivo jedan objekt, njihova agresivna propagacija kroz mrežu uzrokuje pad performansi zvan DoS napad. Primjer takvog napada uzrokovao je *W32/Slammer* [2] računalni crv. Prilikom epidemije ovog malog računalnog crva internetski routeri bili su toliko opterećeni da je 90% poslanih paketa na Internetu nije dostiglo odredište.

Za crv W32/Blaster nagađa se da je uzrokovao nestanak električne energije u SAD-u i Kanadi. Iako mu to nije bila primarna svrha, tom efektu je pridonio usporavanjem komunikacije. Tijekom opće epidemije ovog crva, nije bilo moguće spojiti se ranjivim računalom na nesigurnu mrežu dovoljno dugo da bi se skinule zakrpe koje bi ga onemogućavale. Blaster je pokušao izvršiti napad i na *Windows Update Site* kako bi dodatno onemogućio skidanje zakrpi sa Interneta [21].

2.7.7 Krađa podataka

Napadači se koriste raznim oblicima *malware-a* kako bi došli do privatnih podataka korisnika, poput lozinki ili brojeva kreditnih kartica i na taj način otimali njihov novac.

Računalni crvi često koriste *phishing* napad kao svoj teret. To je napad kojim se korisnika traži određena lozinka (npr. kreditna kartica) pomoću lažnog predstavljanja (*e-mail* poruka koja se čini da je stigla od banke ili kreditne kuće). Unešeni podaci se šalju

napadaču koji ih može koristiti po volji. Osim toga, generiraju se nove e-mail poruke koje se propagiraju novim žrtvama na mreži. Čak 5% žrtava nasjedne na takav napad [2], što s obzirom na broj korisnika Interneta upućuje na velike količine ljudi i velike iznose novaca.

Backdoor svojstva također su čest teret računalnih crva. Iako ne stvaraju neku izravnu štetu, omogućavaju neovlašten pristup napadača računalu i podacima na njemu. Identitet zaraženog računala šalje se primjerice u obliku IP adrese napadaču koji tada može ostvariti pristup.

Keyloggeri također mogu poslužiti kao teret za krađu podataka. Oni pamte znakove utipkane na tipkovnici i šalju ih napadaču.

3. Tehnike napada *malware-a*

3.1 Boot virusi

Većina računala nema vlastiti operacijski sustav već ga mora dohvatiti s diska ili putem mreže. Na osobnim računalima (*IBM PC*) diskovi su organizirani u particije kojima su pridodana određena logička imena (C:, D: na *Windows NT* ili *MS DOS* operacijskim sustavima, „hda1", „hda2" na *Linux* operacijskim sustavima itd.) Osim toga postoji mogućnost posebnih skrivenih particija koje računalo koristi za pohranjivanje dodatnih *BIOS* alata na disku. One su teže dohvatljive jer im se ne pridodaje ime.

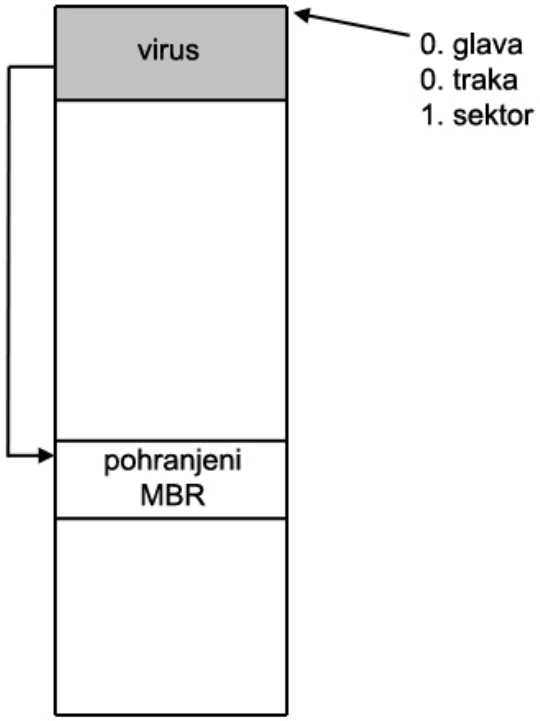
Na starijim računalima se sustav podizao s diskete, i nije nudio mogućnost mijenjanja *boot* poretka. *BIOS* čita prvi sektor *boot* diska, pohranjuje ga u memoriju i izvršava. To je predstavljalo jednostavan način da se maliciozan kod izvrši prije operacijskog sustava.

Novija računala rade na drugačiji način, i omogućavaju da se diskovi dijele na više particija. Disk se dijeli na „glave" (*heads),* „trake" (*tracks*) i „sektore" (*sector*). Glavni zapis za učitavanje(*Master Boot Record – MBR*) se nalazi u glavi 0, traci 0, sektoru 1. Radi se o prvom sektoru tvrdog diska. U *MBR* se nalazi programski kod koji ovisi o procesoru, i koji pronalazi aktivnu *boot* particiju iz zapisa tablice particija (*partition table – PT*). Tablica particija se nalazi u *MBR*. Osim toga, na početku *MBR* se nalazi kratki kod zvan program punilac (*boot strap loader).*

Adresa prvog i zadnjeg sektora u particiji.
Oznaka je li particija *bootabilna*.
Oznaka tipa.
Udaljenost (*offset*) prvog sektora particije od početka diska u sektorima.
Veličina particije u sektorima.

Slika 3.1 *Partition Table*

Loader pronalazi prvi logički sektor aktivne particije (*boot sector*) koji sadrži kod specifičan za operacijski sustav i puni ga u memoriju. Karakteristika koja omogućava napad, je mogućnost da se *MBR*, čiji kod ne ovisi o operacijskom sustavu, zamijeni kodom virusa. Nakon pokretanja virusa, on ostaje aktivan u memoriji i poziva originalni *MBR* nakon sebe.



Slika 3.2 Inficirani *Master Boot Record*

Boot sektor disketa (*floppy disks*) je prvi sektor diskete koji sadrži imena datoteka koje treba pohraniti u memoriju, i koje su specifične za operacijski sustav. Stariji *PC*-ovi su sadržavali manu u dizajnu, koju su autori *boot* virusa iskorištavali. Naime, u slučaju da se disketa nalazila u računalu, *BIOS* bi prvo pokušao podići sustav sa nje.

MBR je veličine 512 byte-ova, što je dovoljno za male viruse. Najčešća tehnika *MBR* infektora napada samo *boot strap* kod, koji se nalazi na početku sektora. *PT* mora ostati netaknuta kako bi *MS DOS* mogao pronaći podatke na disku. Originalni *MBR* se pohranjuje na nekoj daljoj poziciji na disku, kako bi se mogao izvršiti nakon izvršenja virusa. Virusi se često oslanjaju na činjenicu da je nekoliko sektora nakon *MBR* prazno, i time prikladno za pohranu originalnog *MBR*: To ipak nije u svim slučajevima tako, što uzrokuje *nebootabilnost* sustava nakon infekcije.

Neki od boot virusa ne pohranjuju originalni *MBR* već ga samo prepisuju, ostavljajući također *PT* u originalnom stanju. Oni se sami brinu o pronalasku aktivne particije kojoj tada prebacuju kontrolu.

Osim napada na *boot strap loader* očita meta je i tablica particija. Promjenom njenih vrijednosti moguće je lako natjerati sustav da umjesto prvog sektora aktivne particije dohvati sektor zaražen virusom. Virus će nakon izvršavanja opet pozvati izvršavanje originala.

Originalni *MBR* je moguće pohraniti i na kraju tvrdog diska. S obzirom da postoji mogućnost da te lokacije korisnik prepiše svojim podacima, i time učini sustav nesposobnim za podizanje, neki virusi smanjuju veličinu particije kako bi zaštitili taj dio od prepisivanja podataka.

Puno češća metoda od infekcije *MBR* je infekcija *boot* sektora. Obično se radi o infekciji prvog sektora diskete, ali postoje i infekcije tvrdog diska. Prvi sektor se zamjenjuje kodom virusa.

Originalni boot sektor se može pohraniti na kraju *root* direktorija. Slaba strana takvog pohranjivanja je što će u slučaju velikog broja datoteka u *root* direktoriju neka imena biti prepisana, što će rezultirati ispisom besmislenih znakova kod ispisa datoteka. (primjer: Stoned virus na *MS DOS-u*)

Još jedna lokacija za pohranu originalnog *boot* sektora su neiskorišteni klasteri *MS DOS FAT* sustava označeni sa „*BAD*" zastavicom.

Neke vrste disketa sadržavaju područja koja nisu inicijalno dohvatljiva korisniku. Ta područja iskorištavaju virusi koji su preveliki da bi stali u samo jedan sektor. Osim toga takva metoda je otežavala pronalazak i uklanjanje starijim antivirusnim programima.

U nekim slučajevima virusi uopće ne pohranjuju originalni *boot* sektor diskete koju inficiraju. Inficiranjem *MBR* ili *boot* sektora na tvrdom disku, kontrola se prebacuje na pohranjeni *boot* sektor na tvrdom disku. Disketa se teško može vratiti u originalno stanje jer postoji velik broj različitih *boot* sektora (koji ovise o operacijskom sustavu) između kojih se mora izabrati. Osim toga, slično kao i kod infekcije *MBR* neki virusi pokušavaju obaviti i funkcionalnost originalnog *boot* sektora unutar svog malicioznog koda, i dohvatiti potrebne sustavske datoteke.

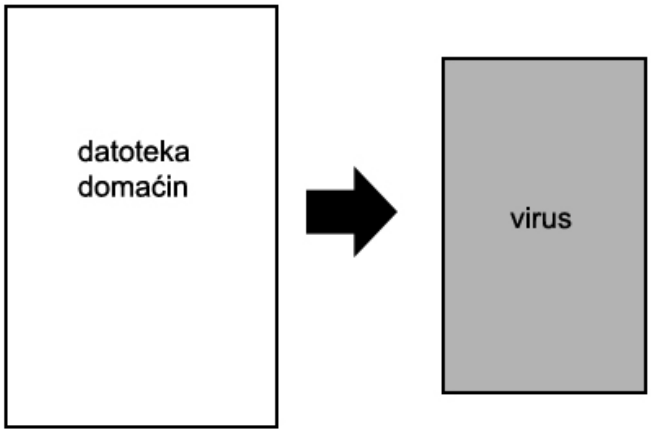
Kao i kod infekcije *MBR boot* sektor se kod nekih virusa pohranjuje na kraj diska ili aktivne particije, a ponekad uz smanjivanje veličine te particije kao oblik zaštite od prepisivanja korisničkim podacima.

Boot virusi se više ne pojavljuju u velikom broju, i većina ih je predviđena za *MS DOS* operacijski sustav. No ipak, postoje i noviji boot virusi, koji napadaju upravljačke programe (*driver)* za upravljanje disketama, na *Windows 95* operacijskom sustavu. Brisanjem tih upravljačkih programa omogućava se *boot* infekcija [2].

3.2 Infekcija datoteka

3.2.1 Virusi koji prepisuju postojeće datoteke

Jedna od najjednostavnijih izvedbi virusa koji napadaju datotke su virusi koji prepisuju postojeće datoteke (*overwriting viruses*). Njihovo širenje s svodi na uništavanje datoteka domaćina i umetanje svojih kopija na njihovo mjesto. Ova tehnika nikako se ne može smatrati sofisticiranom, ali njeni učinci mogu biti vrlo destruktivni. Datoteke inficirane takvim virusom nemoguće je dezinficirati, jer je njihov sadržaj nepovratno izbrisan. Jedina mogućnost za vraćanje je kopiranje iz sigurnosnih kopija (*backup*). Širenje zaraze ovakvim virusima ovisi o dodatnim sposobnostima, poput propagacije kroz mrežu. Ukoliko se virus širi isključivo upisivanjem svog koda preko sadržaja korisnikovih datoteka, nefunkcionalnost sustava će vrlo brzo biti primjećena.



Slika 3.3 Prvi tip *overwriting* virusa

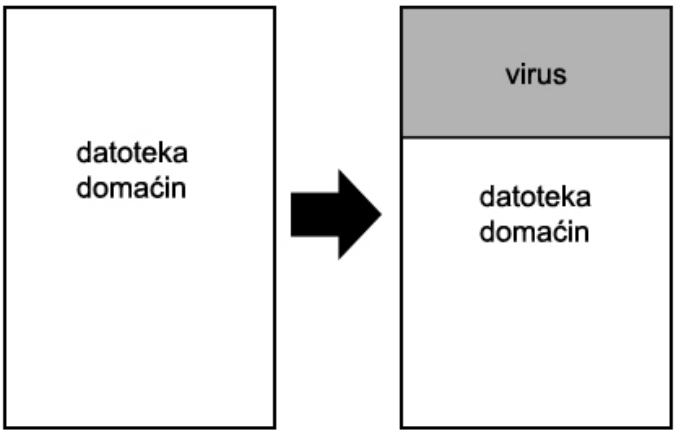
1990-ih godina, autori virusa pokušavali su napisati što kraći mogući binarni (*binary*) virus. Zbog smanjenja veličine, iz njihovog koda su izbacivani mnogi dijelovi, pa čak i oni koji im povećavaju vjerojatnost širenja. Tok takvih virusa sastojao se od tri dijela:

- pronađi bilo koju datoteku domaćina u trenutnom direktoriju
- otvori datoteku za pisanje
- zapiši svoj kod preko sadržaja otvorene datoteke

Kako bi kod virusa bio što manji, izbačena je čak i mogućnost da pronađe iduću datoteku u trenutnom direktoriju. Zbog toga je takav virus najčešće sposoban infcirati samo jednu datoteku unutar jednog direktorija.

Neki virusi iz ove skupine koriste *BIOS* funkcionalnost za pisanje po disku, umjesto funkcija *MS DOS-a*. Takav virus moguće je napisati u čak samo petnaest *byte-ova.* Virus može zapisati svoj kod u svaki sektor diska, ali time čini sustav toliko neupotrebljivim da zaustavlja i vlastito širenje.

Postoji jednostavna mogućnost da *overwriting* virus ne promijeni veličinu inficirane datoteke, tako da zapiše svoj kod na početak te datoteke, a ostatak ostavi netaknut. I takva datoteka će izgubiti originalnu funkcionalnost, ali će „izvana" biti manje uočljiva.



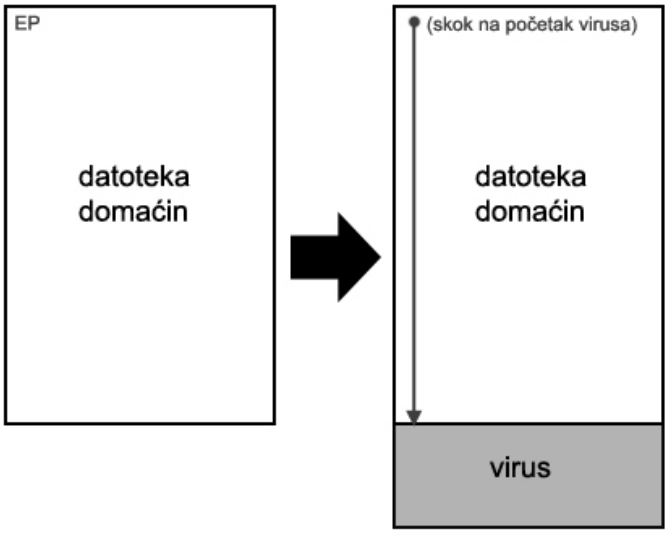
Slika 3.4 Drugi tip *overwriting* virusa

Osim na početku, neki virusi svoj kod zapisuju na proizvoljnu poziciju u datoteci. Veličina koda time ostane nepromjenjena. Izvršavanje datoteke domaćina odvijat će se do trena kada na red za izvršavanje dođe instrukcija koja pripada kodu virusa. Do tog trenutka uopće ne mora doći, tako da je širenje takvog virusa upitno. Najčešći rezultat je pad programa prije izvršenja virusa [2].

3.2.2 Virusi koji svoj kod dodaju na kraj inficiranih

Ovu tehniku može se, uz neke preinake, primijeniti na velik broj različitih izvršnih datoteka, poput *COM*, *MS DOS EXE, NE, PE* formata. Prilikom širenja takvih virusa, tijelo virusa se dodaje na kraj inficirane datoteke (*appending viruses*). U slučaju napada na *COM* datoteke, nekoliko početnih *byte-ova* originalnog izvršnog koda zapisuje se isto na kraj datoteke, kako bi se sačuvali. Na početak datoteke zapisuje se instrukcija za skok na adresu koja je prije označavala kraj datoteke, a sada označava početak koda virusa. Prilikom pokretanja, ta instrukcija predaje kontrolu virusu, koji se širi i eventualno izvršava teret (*payload*). Nakon toga sačuvane instrukcije sa početka zapisuju se u memoriji nazad na njihovo mjesto i virus predaje kontrolu originalnom

programu. Funkcionalnost originalnog programa ostaje sačuvana, što čini virus manje primjetnim i povećava mu vjerojatnost da će se neopaženo širiti.

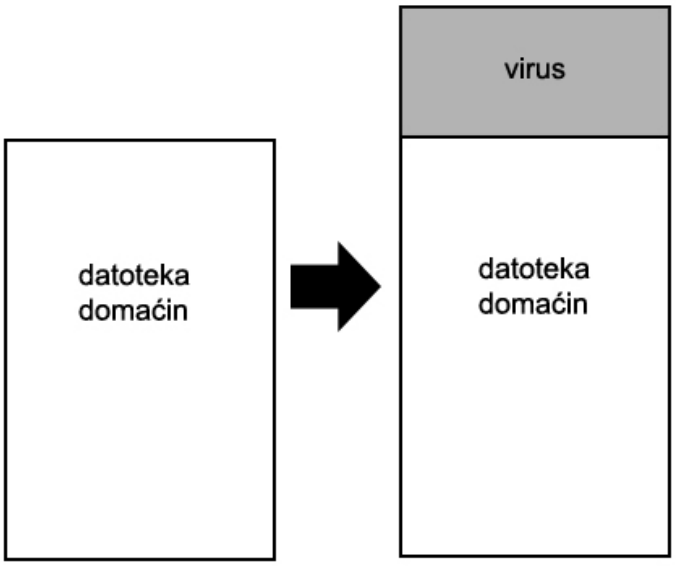


Slika 3.5 *Appending* virus

Napredniji formati izvršnih datoteka poput *PE* imaju zaglavlja u kojima je upisana ulazna točka programa (*entry point*). Pri njihovoj infekciji mijenja se vrijednost *entry point-a* na vrijednost početne adrese koda virusa. Originalni *entry point* se zapisuje na kraju datoteke, kako bi virus mogao nakon izvršenja prenjeti kontrolu originalnom programu [2].

3.2.3 Virusi koji svoj kod dodaju na početak inficiranih datoteka

Ova tehnika infekcije je jednostavna, i korištena je u velikom broju virusa. Princip širenja takvog virusa je dodavanje tijela virusa ispred koda originalne datoteke domaćina (*prepending viruses*). Na taj način pokretanjem inficirane datoteke virus odmah dobiva kontrolu. Kako bi se nakon izvršenja virusa kontrola predala originalnom programu, potrebno je znati odgovarajuću adresu. Kod *COM* formata to je jednostavno, jer se ta adresa nalazi točno iza tijela virusa, a veličina virusa poznata je njegovom autoru. Napredniji formati izvršnih datoteka nastaju obično kompiliranjem iz viših programskih jezika, pa je teže odrediti *entry point,* tj. adresu na kojoj se kontrola predaje originalnom programu

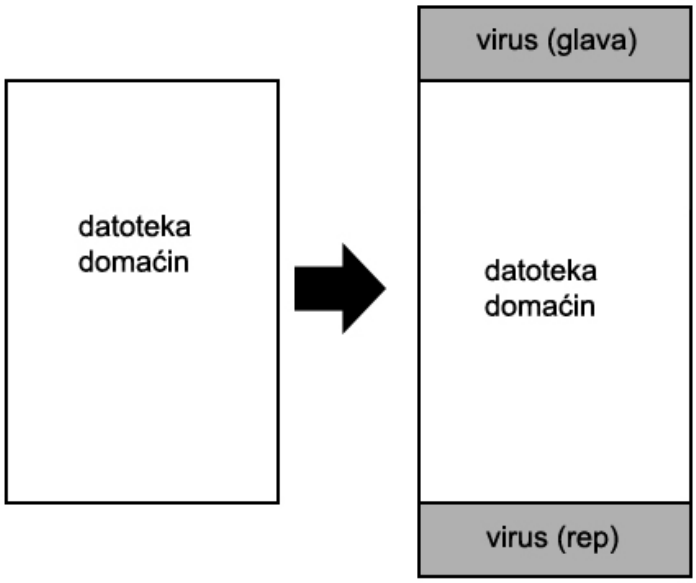


Slika 3.6 *Prepending* virus

Podskupina *prepending* virusa su parazitski virusi (*parasitic viruses*). Njihova tehnika napada se također zasniva na ubacivanju tijela virusa na početak, ali tako da se zapiše preko originalnog programa. Prepisani dio originala zapisuje se na kraj datoteke. Nakon izvršenja virusa kontrola se prebacuje na adresu koja je jednaka veličini neinficirane datoteke gdje se nalazi početak originalnog programa. Neki parazitski virusi koriste dodatne datoteke za pohranjivanje početka originalnog programa koji inficiraju. Te su datoteke obično skrivene (imaju *hidden* atribute na *DOS-u*) [2].

3.2.4 Ameboidni virusi

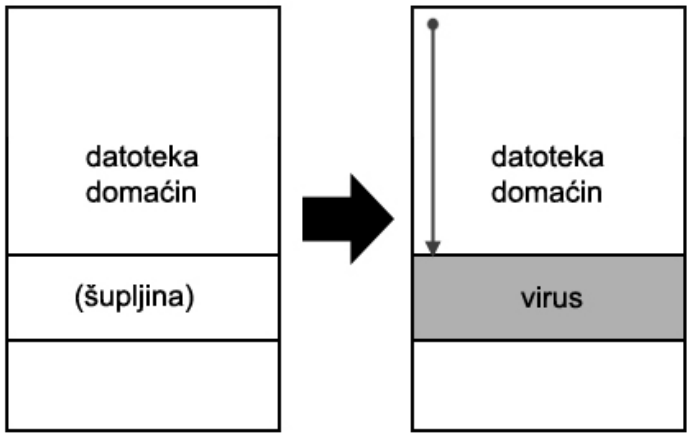
Ovaj oblik virusa obuhvaća tijelo originalnog programa unutar dva dijela koda virusa. Virus se sastoji od glave i repa (*head,tail*) između kojih se nalazi originalni kod (*amoeba infection technique*). Pri pokretanju se originalni program pohranjuje u zasebnu datoteku kako bi se nakon izvršavanja virusa mogao pokrenuti. Ova tehnika se rijetko susreće u stvarnim virusima [2].



Slika 3.7 Ameboidni virus

3.2.5 Virusi koji popunjavaju „šupljine” u datotekama

Oblik binarnih virusa koji ne mijenjaju veličinu inficirane datoteke su virusi koji ispunjavaju „šupljine“ u datotekama (*cavity viruses*). Oni koriste dijelove datoteke u koje sigurno mogu zapisati svoj kod, bez da naruše stabilnost originalnog programa. Kod *COM* formata to su primjerice dijelovi ispunjeni nulama, ili znakovima praznine. U naprednijim formatima poput *PE, cavity* virusi koriste sekcije premještanja (*relocations sections*) koje se pri normalnom izvršavanju obično ne koriste.Ovakav pristup je ograničen veličinama „šupljina” u datoteci, i zbog toga su *cavity* virusi najčešće vrlo spori infektori.

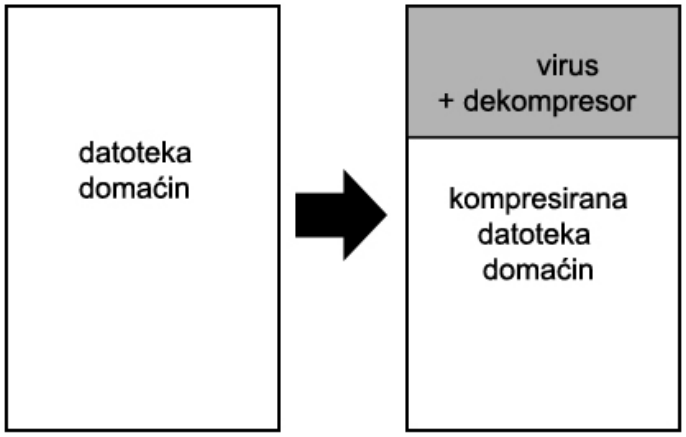


Slika 3.8 *Cavity* virus

Napredan oblik ove tehnike napada je razdijeljeno traženje „šupljina” (*fractionated cavity viruses*). Takvi virusi se sastoje od glave programa i više komadića koda koji se smješta u šupljine. Glava virusa pronalazi dijelove koda pomoću vlastite tablice udaljenosti i pohranjuje ih u kontinuiran niz u memoriji. *Entry point* programa se na neki od opisanih načina mijenja na početak virusa. Ovom tehnikom se ne povećava veličina originalne datoteke domaćina, ali ovisnost o veličini „šupljina” u datoteci domaćinu je manja [2].

3.2.6 Sažimajući virusi

Poseban oblik virusa su sažimajući virusi (*compressing viruses*). Oni koriste neki od algoritama kompresije kako bi saželi tijelo napadnute datoteke. Razlog tome može biti skrivanje promjene veličine datoteke. Najčešće takvi virusi nemaju u svome kodu implementirane rutine za kompresiju i dekompresiju, već se koriste funkcionalnošću okruženja, kao što su ugrađeni (*runtime*) programi za kompresiju. Pri pokretanju zaražene izvršne datoteke, kontrola se prebacuje kodu virusa, koji se brine za otpakiravanje tijela originalnog programa. Ovom tehnikom napada moguće je izbjeći promjenu veličine, te promjenu funkcionalnosti originalnog programa [2].

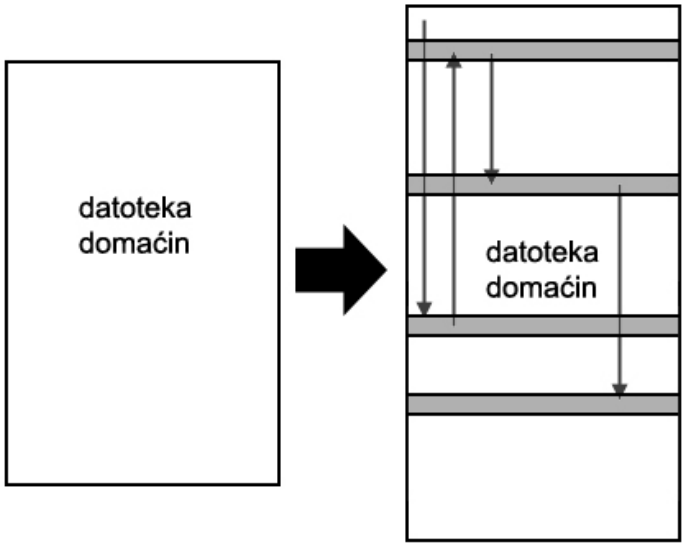


Slika 3.9 Sažimajući virus

3.2.7 Tehnika komadanja tijela virusa

Ovdje se radi o složenoj tehnici napada koja dijeli tijelo virusa u više manjih dijelova (*virus body tehnique*). Ti dijelovi se razmještaju na slučajna mjesta u datoteci te prepisuju originalni kod. Uništeni dijelovi se pohranjuju na kraju programa kako bi funkcionalnost originala ostala netaknuta. Kod pokretanja, kontrola se predaje prvom dijeliću virusa, koji onda predaje kontrolu idućem, i tako dalje, sve do glavnog tijela

virusa. Ovakvi virusi su posebno otporni na otkrivanje jer je potrebno pratiti tok programa kako bi se pronašli svi dijelovi virusa. Dezinfekcija također predstavlja teškoće zbog slučajnog odabira koji će se dio originalnog koda zamijeniti komadićem virusa [2].



Slika 3.10 *Virus body tehnique*

3.3 WIN32 virusi

3.3.1 Win32

Pojavom operacijskog sustava *Windows 95* napravljene su velike promjene u odnosu na postojeće operacijske sustave poput *MS DOS-a*. *Windows 95* više nije dopuštao korištenje nekih funkcionalnosti *MS DOS-a* koje su olakšavale rad virusima. Taj sustav je bio nov i nepoznat većini autora *malware-a*, koji su stoga prvo pokušavali postojeće viruse prilagoditi tako da se mogu izvršavati i na novom operacijskom sustavu. *Windows 95* uvode pojam *Win32* koji označava skupinu funkcija tj. sučelja prema funkcijama operacijskog sustava (*application programming interface – API*). Te iste *API* funkcije se koriste i u *Windows* operacijskim sustavima stvorenima kasnije, uključujući *Windows 9x, Windows NT, Windows XP itd.* Unutarnja implementacija *API* funkcija nije poznata, ali većina njih se može pozvati sa raznih *Windows* operacijskih sustava. Razlike u dostupnim funkcijama obično ovise o platformi na kojoj operacijski sustav radi i o dostupnom sklopovlju.

Većina virusa napisanih za operacijske sustave nakon uvođenja *Win32* oslanjaju se na *Win32 API* što ih čini prenosivim između različitih inačica 32-bitnih *Windows* operacijskih sustava. Daljnju pogodnost za prenosivost takvog *malware-a* pruža i *PE* format izvršnih datoteka.

3.3.2 PE format izvršnih datoteka

PE (*portable executable*) format je stvoren kao dio *Win32* specifikacije. Predviđen je za sve operacijske sustave koji koriste *Win32*, a pretpostavlja se da će biti korišten i u nadolazećim generacijama *Windows* operacijskih sustava. Format je izveden iz prijašnjeg *COFF* (*common object file format*) formata. „Portabilnost" u nazivu formata je stavljena jer je format predviđen za razne *Win32* sustave. *COFF* i *PE* sadržavaju velik dio zajedničkih struktura. *COFF* se čak koristi kao format *OBJ* datoteka koje stvaraju *Microsoft-ovi* kompilatori u procesu kompilacije.

Pojavom 64-bitnih *Windows-a PE* format je ostao najvećim dijelom nepromijenjen. Izbačeno je jedno polje, a neka su modificirana tako da umjesto 32 imaju 64 bita.

PE format se koristi za više vrsta datoteka. *EXE* i *DLL* datoteke koriste isti format. Razlika je u samo jednom bitu koji označava treba li se datoteka smatrati kao *EXE* ili *DLL*. *DLL* datoteke mogu imati razne ekstenzije, poput *.OCX*.

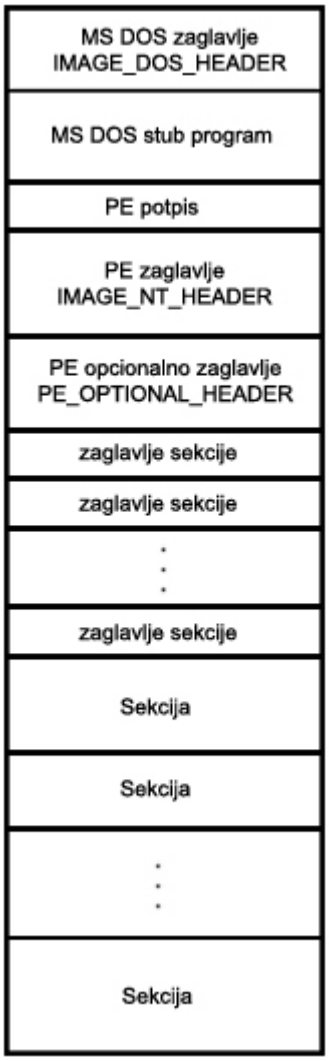
Izgled *PE* datoteka dokumentiran je, i dostupan na više mjesta, ali najdoslovniji prikaz je zapisan u datoteci *WINNT.H*, koja sadrži strukture zaglavlja i definicije potrebne za programe koji barataju *PE* datotekama.

Svaka *PE* datoteka počinje malim *MS DOS EXE* programom. Razlog tome je iz vremena kada je malo ljudi koristilo *Windows* operacijski sustav. Taj program ispisuje da su potrebni *Windows-i* za izvođenje ovog programa. Na samom početku se nalazi *MS DOS* zaglavlje, nakon čega slijedi kratki kod programa (*stub program*) koji ispisuje navedenu poruku. Zaglavlje sadržava više polja, od kojih su za *PE* datoteke dva značajnija. Prvo polje se zove *e_magic* koje sadržava „magičnu" vrijednost 0x5A4D. U *ASCII* vrijednostima to označava slova „MZ" što su inicijali Marka Zibkowskog, jednog od autora *MS DOS-a.* Drugo polje se zove *e_lfanew* i sadržava udaljenost *PE* zaglavlja od početka datoteke. U *WINNT.H* struktura ovog zaglavlja se naziva *IMAGE_DOS_HEADER.*

IMAGE_NT_HEADER je struktura *PE* zaglavlja. Razlike između 32 i 64-bitnih struktura su zanemarive za opći opis. Zaglavlje se sastoji od tri dijela. Prvi je „*PE* potpis" koji sadržava vrijednost 0x00004550, tj. u *ASCII* vrijednosti „PE00". Slijedi *IMAGE_FILE_HEADER*, dio zaglavlja koji sadržava neke općenite podatke o datoteci, od kojih je najbitnija veličina idućeg dijela zaglavlja. Idući dio se zove *IMAGE_OPTIONAL_HEADER.* Iako mu naziv govori drugačije, ovaj dio zaglavlja nije opcionalan.

Iza navedene strukture slijedi tablica sekcija. Radi se o nizu struktura *IMAGE_SECTION_HEADER* koje opisuju svojstva sekcija u *PE* datoteci. Sekcije predstavljaju kod ili neku vrstu podataka. Postoji samo jedna vrsta koda (*binary code*), dok postoji više vrsta podataka. Sekcije sa podacima mogu sadržavati obične programske podatke poput varijabli i konstanti, ali i druge vrijednosti poput

uvoznih(*import)* i izvoznih (*export*) funkcija i slično. Svaka sekcija ima atribute koji označavaju je li sekcija samo za čitanje (*read-only*), za čitanje i pisanje ili zajednička za sve procese koji koriste tu izvršnu datoteku. Minimalno postoje dvije sekcije u *PE* datoteci, jedna za kod i jedna za podatke, no u stvarnosti najčešće postoji minimalno još jedna sekcija. Svaka sekcija ima tekstualno ime koja pomaže pri razlikovanju, ali imena nisu strogo definirana. Tako za sekciju koja sadržava kod *Microsoft* koristi „*.text*" dok *Borland-ovi* kompilatori koriste ime „*CODE*" [4].



Slika 3.11 PE format izvršnih datoteka

Tablica 3.1 *IMAGE_FILE_HEADER*

Veličina	Ime	Opis
WORD	Machine	Ciljana centralna procesorska jedinica za ovaj program
WORD	NumberOfSections	Pokazuje koliko sekcija postoji u tablici sekcija.
DWORD	TimeDateStamp	Sadržava broj sekundi od 1.1.1970 do trenutka kad je datoteka stvorena. Ta vrijednost je točniji podatak od oznake datotečnog sustava.
DWORD	PointerToSymbolTable	Udaljenost tablice COFF simbola od početka datoteke. COFF simboli se rijetko koriste u izvršnim datotekama u novije vrijeme, a opisani su u *Microsoft-ovoj* dokumentaciji.
DWORD	NumberOfSymbols	Broj COFF simbola ako postoje.
WORD	SizeOfOptionalHeader	Veličina strukture koja slijedi *IMAGE_FILE_HEADER,* u slučaju *PE* datoteka to je *IMAGE_OPTIONAL_HEADER.*
WORD	Characteristics	Skup zastavica koje opisuju karakteristike datoteke. Moguće vrijednosti ovog polja mogu se pronaći u *winnt.h* datoteci.

Tablica 3.2 *IMAGE_OPTIONAL_HEADER*

Veličina	Ime	Opis
WORD	Magic	Potpis koji opisuje o kakvom se zaglavlju radi, dvije najčešće vrijednosti su: IMAGE_NT_OPTIONAL_HDR32_MAGIC i IMAGE_NT_OPTIONAL_HDR64_MAGIC
BYTE	MajorLinkerVersion	Glavna inačicu *linkera* koji je korišten za izgradnju izvršne datoteke. Za datoteke stvorene *Microsoft-ovim* linkerom, ovaj broj odgovara inačici programa *Visual Studio*
BYTE	MinorLinkerVersion	Sekundarna inačica *linkera*.
DWORD	SizeOfCode	Ukupna veličina svih sekcija sa IMAGE_SCN_CNT_CODE atributom.
DWORD	SizeOfInitializedData	Ukupna svih sekcija sa inicijaliziranim podacima.
DWORD	SizeOfUninitializedData	Veličina svih sekcija sa neinicijaliziranim podatkovnim atriburima. Ovo polje je često 0 jer linker može dodati neinicijalizirane podatke na kraj regularnih podatkovnih sekcija.
DWORD	AddressOfEntryPoint	Relativna adresa (*Relative Virtual Address – RVA*) prvog *byte-a* koji će se izvršiti. U DLL datotekama može biti

		postavljeno na 0.
DWORD	BaseOfCode	*RVA* prvog *byte-a* izvršnog koda koji se napuni u memoriju.
DWORD	BaseOfData	*RVA* prvog *byte-a* podataka koji se puni u memoriju. Vrijednosti ovog polja nisu postojane kroz različite verzije linkera. (izbačeno u 64-bitnoj verziji)
DWORD	ImageBase	Preferirana adresa za punjenje datoteke u memoriju. Program punioc pokušava ubaciti datoteku na navedenu adresu ukoliko je to moguće.
DWORD	SectionAlignment	Veličina na čiji se višekratnik poravnavaju sekcije kada se pune u memoriju. Poravnanje mora biti veće od poravnjanja u datoteci. Minimum poravnanja je stranica (4K)
DWORD	FileAlignment	Poravnanje sekcija unutar PE datoteke. Za x86 izvršne datoteke, to je 0x200 ili 0x1000.
WORD	MajorOperatingSystemVersion	Glavni broj inačice zahtijevanog operacijskog sustava.
WORD	MinorOperatingSystemVersion	Sekundarni broj inačica zahtijevanog operacijskog sustava.
WORD	MajorImageVersion	Glavni broj inačice ove datoteke. Sustav ju ne koristi i može biti postavljena na 0.
WORD	MinorImageVersion	Sekundarni broj inačice ove datoteke.
WORD	MajorSubsystemVersion	Glavni broj inačice operacijskog podsustava potrebnog za ovu datoteku.
WORD	MinorSubsystemVersion	Sekundarni broj inačice operacijskog podsustava potrebnog za ovu datoteku.
DWORD	Win32VersionValue	Nekorišteno polje, obično postavljeno na 0.
DWORD	SizeOfImage	Sadrži *RVA* koja bi bila pridjeljena adresi sekcije koja bi slijedila zadnju postojeću sekciju. To je količina memorije koju OS mora zauzeti kod punjenja datoteke u memoriju. Ovo polje mora biti višekratnik poravnanja sekcija.
DWORD	SizeOfHeaders	The combined size of the MS-DOS header, PE headers, and section table. All of these items will occur before any code or data sections in the PE file. The value of this field is rounded up to a multiple of the file alignment.
DWORD	CheckSum	Zaštitna suma datoteke. *API* za računanje ove sume CheckSumMappedFile() nalazi se u IMAGEHLP.DLL datoteci.
WORD	Subsystem	Enumeracijska vrijednost koja pokazuje koji podsustav izvršna datoteka očekuje.

WORD	DllCharacteristics	Zastavice koje sadržavaju karakteristike DLL datoteka.
DWORD	SizeOfStackReserve	U EXE datotekama, maksimalna veličina na koji dretva može narasti. Pretpostavljena vrijednost je 1 MB.
DWORD	SizeOfStackCommit	U EXE datotekama, količina memorije koja se zauzima za stog. Pretpostavljena vrijednost je 4 KB.
DWORD	SizeOfHeapReserve	U EXE datotekama, inicijalno rezervirana veličina *heap-a*. Pretpostavljena vrijednost ovog polja je 1 MB.
DWORD	SizeOfHeapCommit	U EXE datotekama, inicijalno zauzeta veličina *heap-a*. Pretpostavljena vrijednost je 4 KB.
DWORD	LoaderFlags	Ne koristi se.
DWORD	NumberOfRvaAndSizes	Na kraju IMAGE_NT_HEADERS je polje struktura IMAGE_DATA_DIRECTORY. Ovo polje sadrži broj unešenih struktura.
IMAGE_	DataDirectory[16]	Niz Struktura IMAGE_DATA_DIRECTORY. Svaka struktura sadrži RVA i veličinu nekog bitnog dijela izvršne datoteke (uvoz, izvoz...)

Tablica 3.3 *IMAGE_SECTION_HEADER*

Veličina	**Ime**	**Opis**
BYTE	Name[8]	*ASCII* ime sekcije. Ne završava nužno sa nulom.
DWORD	Misc.VirtualSize	Sadržava veličinu stvarno iskorištenog dijela sekcije.
DWORD	VirtualAddress	*RVA* gdje sekcija počinje u memoriji.
DWORD	SizeOfRawData	Veličina sekcije u *byte-ovima*. Ukoliko je 0 onda se radi o neinicijaliziranim podacima.
DWORD	PointerToRawData	Udaljenost od početka datoteke gdje počinje sekcija. Mora biti višekratnik poravnanja datoteke.
DWORD	PointerToRelocations	Udaljenost relokacija za ovu sekciju od početka datoteke.
DWORD	PointerToLineNumbers	Udaljenost za *COFF* brojeve linija od početka datoteke za ovu sekciju.
WORD	NumberOfRelocations	Broj relokacija na koje se pokazuje *PointerToRelocations* poljem.
WORD	NumberOfLineNumbers	Broj brojeva linija na koje se pokazuje *PointerToLineNumbers* poljem.
DWORD	Characteristics	Karakteristike sekcije između kojih je izvršena bitovna operacija ILI. Vrijednosti se mogu vidjeti u *WINNT.H* datoteci.

3.3.3 Infekcija zaglavlja

Ovaj tip virusa umeće svoj kod između *PE* zaglavlja i prve sekcije. Vrijednost polja *AddressOfEntryPoint* se mijenja na adresu tijela virusa, a stara vrijednost se pohranjuje kako bi se kontrola vratila originalnom programu. Glavno ograničenje ovakve metode je veličina virusa. Sekcije moraju počinjati na adresi koja je višekratnik polja *FileAlignment.* Stoga, virus može imati maksimalnu veličinu jednaku veličini u *FileAlignment.* Ovu tehniku napada omogućava svojstvo operacijskog sustava (npr. *Windows 95*) koji uredno pokreće takvu datoteku makar ulazna točka ne pokazuje ni u jednu sekciju [2].

3.3.4 *Prepending* PE virusi

Ova metoda vrlo je slična infekciji *MS DOS EXE* datoteka. Virus svoje tijelo postavlja na početak inficirane datoteke. Tada se na početku nalaze zaglavlja virusa. Da bi se kontrola vratila inficiranoj datoteci, stvara se privremena datoteka koja sadržava originalni programi pokreće se. Dezinfekcija ovakvih virusa vrlo je jednostavna jer se ne uništava originalna datoteka [2].

3.3.5 *Appending* PE virusi

Appending virusi koji dodaju nove sekcije

Jedan od načina infekcije *PE* datoteke domaćina je dodavanje novih sekcija u datoteku. Tijelo virusa stavlja se u novu sekciju, i dodaje se zapis u tablicu sekcija. Pri infekciji neki virusi ne promijene polja koja se odnose na veličinu datoteke, veličinu koda, broj sekcija itd. što se može iskoristiti za njegovo otkrivanje.

Appending virusi koji ne dodaju novo zaglavlje sekcije

Kod ovog tipa virusa uočava se malo naprednija metoda infekcije. Virus pri samoreplikaciji ne mijenja broj sekcija u datoteci domaćinu. Umjesto toga, polja zadnjeg zaglavlja sekcije se modificiraju tako da tijelo virusa stane unutar zadnje sekcije. To znači promjenu *VirtualSize* i *SizeOfRawData* polja. Ulazna točka (*AddressOfEntryPoint*) se mijenja tako da pokazuje na početak koda virusa. Stara ulazna točka se naravno pohranjuje radi vraćanja kontrole.

3.3.6 Infekcija *kernel32.dll* datoteke.

Kernel32.dll je dinamička biblioteka (*dynamic link library*) u *MS Windows* okruženju koja se brine o upravljanju memorijom, ulaznim i izlaznim operacijama i prekidima. Prilikom podizanja (*boot*) *Windows* operacijskog sustava, ta datoteka se unosi u adresni prostor sustava, koji je zaštićen od ostalih aplikacija.

DLL datoteke ne koriste se kao programi u smislu da imaju jednu ulaznu točku iz koje se pokreću. Zbog toga virusi koji inficiraju *kernel32.dll* ne napadaju *AddressOfEntryPoint* polje zaglavlja. Kako je ideja te datoteke da izvozi funkcije (*API-je*) u njoj postoje zapisi izvoznih (*export*) adresa na kojima se nalaze pojedine funkcije. Zapravo se radi o ulaznim točkama pojedine izvozne funkcije. Zbog toga, jednostavan način infekcije je promjena ulazne točke *API* funkcija na adresu virusnog koda pridodanog na kraju datoteke. Na taj način virus dolazi do kontrole, a vraćanje kontrole lako se obavlja pohranom originalne ulazne točke za funkciju. Sustav pri podizanju unosi tu datoteku u memoriju i pri svakom pozivu određene *API* funkcije virus dobiva kontrolu.

Ipak, postoji razlika između verzija *Windows* operacijskog sustava, što se tiče ranjivosti na ovaj napad. *Windows NT* (za razliku od *Windows 95*) kod svakog unošenja *DLL* datoteke u memoriju provjerava zaštitnu sumu zapisanu u jedno od polja zaglavlja *PE* datoteke. Ako to polje ne odgovara stvarnoj vrijednosti nanovo izračunate sume, dojavljuje se greška. Izračunavanje sume nije dokumentirano, ali postoje *API* funkcije (*CheckSumMappedFile*() unutar *IMAGEHLP.DLL* biblioteke) koje to omogućavaju. Zbog toga je infekcija na novijim *Windows* inačicama teža, ali ne i nemoguća [2].

3.3.7 Virusi pratioci

Ova skupina virusa nije jako česta. Oni se pouzdaju na jednostavnu činjenicu da će u slučaju postojanja dvije datoteke u istom direktoriju s istim imenom, od kojih jedna ima *COM* a druga *EXE* ekstenziju, operacijski sustav prvo izvršiti onu sa *COM* ekstenzijom. Ovakvi virusi traže *EXE* izvršne datoteke na disku i samorepliciraju sebe u isti direktorij (ili u put izvršavanja – *path*) sa istim imenom i *COM* ekstenzijom.

3.3.8 *Fractionated cavity* virusi na *Win32*

Prema strukturi *PE* datoteka, sekcije moraju biti poravnate prema veličini zadanoj u *FileAlignment* polju u zaglavlju datoteke. Iz tog razloga, između popunjenih dijelova sekcija najčešće postoje prazni prostori, obično popunjeni nulama. Inicijalna (*default*) veličina postavljena u *FileAlignment* polje je 512 *byte-ova*, što je standardna veličina sektora na tvrdom disku. Iz toga slijedi da su prazni dijelovi sekcija manji ili jednaki 512 *byte-ova*. Zbog toga, *PE* virusi najčešće moraju koristiti *fractionated cavity* metodu za infekciju *PE* datoteka. Tijelo virusa se dijeli u manje segmente kojima se popunjavaju slobodni prostori. Dio koda virusa zadužen je za punjenje koda u jedan kontinuirani prostor alocirane memorije.

Prednost ovakvih virusa je što ne mijenja veličinu datoteke čime postaje manje uočljiv. Druga prednost je nepoznata udaljenost dijelova koda virusa od početka datoteke. Ta činjenica uvelike otežava postupak antivirusnim alatima da točno prepoznaju virusni kod.

3.3.9 Napad na *lfanew* polje u zaglavlju

Mnogi virusi koriste ovu tehniku napada zbog njene jednostavnosti. Virus je također *PE* datoteka, i cijeli njen sadržaj se dodaje na kraj datoteke domaćina. U *DOS* zaglavlju virus mijenja polje *lfanew* na adresu *PE* zaglavlja virusa. Virus dalje radi kao normalan program za *Windows* operacijski sustav, a nakon izvršenja stvara privremenu datoteku koja je jednaka originalnoj datoteci domaćinu, sa originalnom vrijednosti *lfanew* polja. Virus pokreće tu privremenu datoteku kako bi se izvršio originalni program.

3.3.10 Virusi koji napadaju *VxD upravljačke programe*

Na *DOS* operacijskom sustavu programi su direktno pristupali pojedinom sklopovlju, hvatajući prekide i koristeći memoriju uređaja. Aplikacija je tad imala potpunu i isključivu kontrolu nad uređajem. U kasnijim operacijskim sustavima (do *Windows 95* uključivo) više aplikacija ima mogućnost simultanog rada. Kako bi se omogućilo tim aplikacijama da koriste isto sklopovlje, uvedeni su virtualni upravljački programi na *Windows 3.0*. Umjesto direktne komunikacije sa uređajem, aplikacija komunicira sa virtualnim uređajem kojeg kontrolira operacijski sustav. Ti upravljački programi se dinamički pune u memoriju, i kontroliraju eventualne sukobe aplikacija nad zajedničkim sklopovljem. Ime *VxD* (*virtual device driver*) je nastalo jer se većina tih upravljačkih programa nazivala V<neka oznaka>D.386.

U *Windows* okruženju definirana je sigurnosna hijerarhija nazivana „prstenovi" (*rings*). Prsten označava razinu tj. sloj sigurnosti na kojem neka aplikacija radi. Broj prstena označava razinu privilegija, a prsten broj 0 se obično uzima kao razina sa najviše mogućih privilegija. Ovaj sustav često je ostvaren sklopovski unutar nekih *CPU* arhitektura.

U *Windows* okruženju (*Windows 95* operacijski sustavi i noviji), obične aplikacije rade u zaštićenom načinu rada, točnije rečeno, na prstenu broj 3. To znači da imaju ograničen pristup memoriji i hardveru. Za razliku od toga *VxD driveri* rade na prstenu broj 0 koji ima najvišu moguću kontrolu. Zbog toga aplikacije koje trebaju veću kontrolu (uključujući i antivirusne programe) instaliravaju *VxD* drivere na sustav. Nažalost iz istog razloga su *VxD* driveri meta virusa koji žele maksimalnu kontrolu nad sustavom.

Virusi ove vrste traže *drivere* sustava (najčešće one koji su poznati, tj. standardni) i inficiraju ih svojim kodom. Pri slijedećem podizanju sustava ti *driveri* biti će napunjeni u memoriju, i radom u prstenu 0 imat će pristup svim dijelovima sustava.

Na novijim *Windows* sustavima (98,2000…) *VxD driveri* više nisu podržani nego ih je zamijenio *WDM(Windows driver model)*. Zbog toga virusi koji su iskoristavali ovu tehniku na *Windows 95* operacijskom sustavu neće biti kompatibilni sa novijim verzijama [2].

3.4 In-Memory tehnike napada

3.4.1 Direktno aktivni virusi

Ova katogorija virusa odnosi se na jednostavnije viruse koji borave u memoriji samo kratko vrijeme tijekom pokretanja (*direct-action viruses*). Pokretanjem zaražene datoteke virus dolazi u memoriju i traži nove datoteke domaćine kako bi ih zarazio. Infekcija većeg broja datoteka traje duže i vjerojatnije je da će se otkriti, stoga najčešće takvi virusi inficiraju mali broj datoteka pri svakom pokretanju. Radi se o sporim infektorima, osim u slučajevima kada imaju sposobnost širenja preko mreže.

Nakon izvršavanja koda virusa, on nestaje iz memorije sve do idućeg pokretanja neke od zaraženih datoteka. Stvaranje ovakvih virusa je manje zahtjevno, ali je njihova sposobnost djelovanja manja.

3.4.2 Virusi rezidentni u memoriji *MS DOS* okruženju

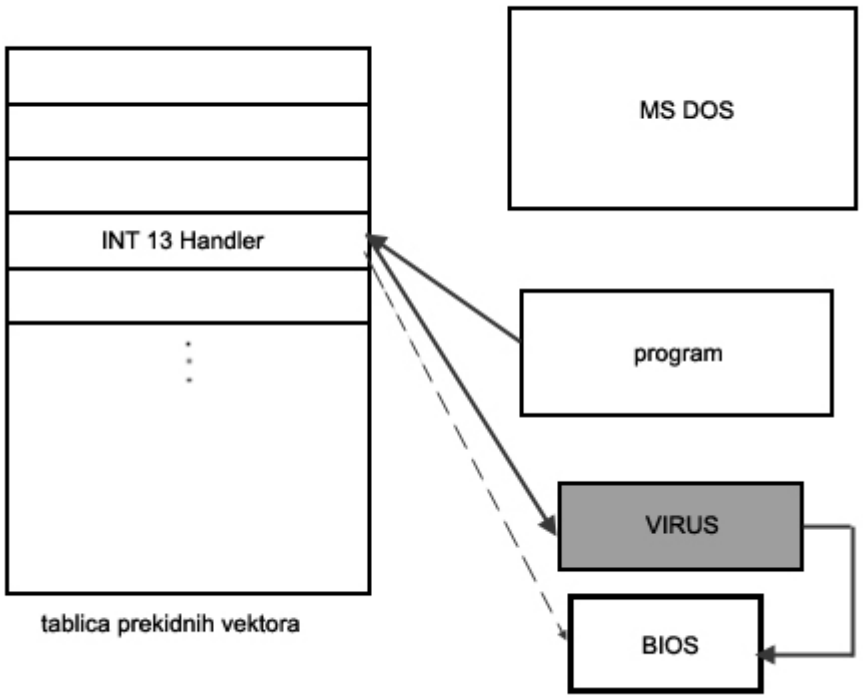
Ova naprednija tehnika napada virusa ima neke uobičajene korake koji se koriste kako bi virus ostao rezidentan u memoriji (*memory-resident viruses*). Time se postiže da nije nužno svaki put pokrenuti inficiranu datoteku kako bi se inficirale nove datoteke domaćini. Ti koraci su:

- Virus dobiva kontrolu nad sustavom.
- Virus alocira dio memorije
- Virus puni alociranu memoriju svojim kodom.
- Virus aktivira sebe u alociranom bloku memorije.
- Virus „zakači" (*hook*) svoj kod na tok izvršavanja koda.
- Slijedi infekcija novih datoteka ili dijelova sustava.

Na operacijskim sustavima koji ne podržavaju simultano izvođenje više programa, svaki program koji dobije kontrolu, u tom trenu ima isključivu kontrolu nad sustavom. To znači da se nijedan drugi program pri tome ne može izvršavati. Jedan od takvih sustava je i *MS DOS*. Iz tog razloga *memory-resident* virusi na tom operacijskom sustavu ne mogu nastaviti raditi nakon prvotnog izvršavanja. Umjesto toga, koristi se metoda „*terminate and stay resident"*. To znači da virus kopira svoj kod negdje u memoriju, ali nakon izvršavanja vraća kontrolu operacijskom sustavu ili drugom programu. *MS DOS* koristi posluživanje prekida, što se može iskoristiti za implementaciju ovakvog napada.

U *MS DOS* okruženju programi koriste prekide kako bi koristili sustavske servise. Za razliku od starijih sustava, adrese potprograma za posluživanje prekida ne moraju biti poznate programeru, već se koristi tablica prekidnih vektora u kojima su zapisane sve potrebne adrese. Tako je moguće pozivati prekidne potprograme samo pomoću prekidnog broja (naredba *INT <broj>*). Tablica prekidnih vektora nalazi se na početku fizičke memorije. Adresa prekidnog potprograma lako se može zamijeniti u

tablici prekidnih vektora tako da pokazuje na tijelo virusa. Tako *boot* virusi često koriste *INT 13* prekidni potprogram za pristupanje disku. Tako virus može preuzeti kontrolu pri svakom pristupu disketi, i na taj način se proširiti na nju [2].



Slika 3.12 Virus rezidentan u memoriji

3.4.3 *Swapping* virusi

Metoda vrlo slična onoj koju koriste *memory resident* virusi naziva se *swapping* tj. zamjena podataka. Takvi virusi se također pomoću malog dijela koda zakače na neki događaj poput prekidnog potprograma. Tek okidanjem tog događaja taj mali dio koda u memoriju puni tijelo virusa sa diska i inficira novu datoteku, ili dio sustava.

Prednosti ove metode su zauzimanje manje fizičke memorije. Osim toga, tijelo virusa moguće je skrivati od antivirusnih alata, npr. kriptiranog na tvrdom disku. No unatoč tome, često čitanje sa diska troši određeno vrijeme, stoga je jednostavnije uočiti neuobičajeno ponašanje sustava [2].

3.4.4 Virusi u korisničkom načinu rada (*Win32*)

Noviji operacijski sustavi podrazumijevaju simultano izvođenje više zadataka (*multitasking*). Zbog toga virusi pisani za tu platformu ne koriste starije metode kako bi

ostali rezidentni u memoriji. Pristupi kako to napraviti su raznoliki. Za razliku od *DOS-a,* na sustavima baziranim na *Windows NT* tehnologiji, programi koji rade u korisničkom načinu rada ne mogu koristiti memoriju sustavskih procesa, tj. onih koji rade u jezgrenom načinu rada. Time se postiže veća stabilnost i sigurnost sustava.

I *Windows* virusi se koriste *direct-action* tehnikom kako bi inficirali datoteke domaćine. Virus preuzima kontrolu u okviru originalnog procesa. Tada može stvoriti vlastitu dretvu unutar procesa koja će inficirati slijedeće objekte.

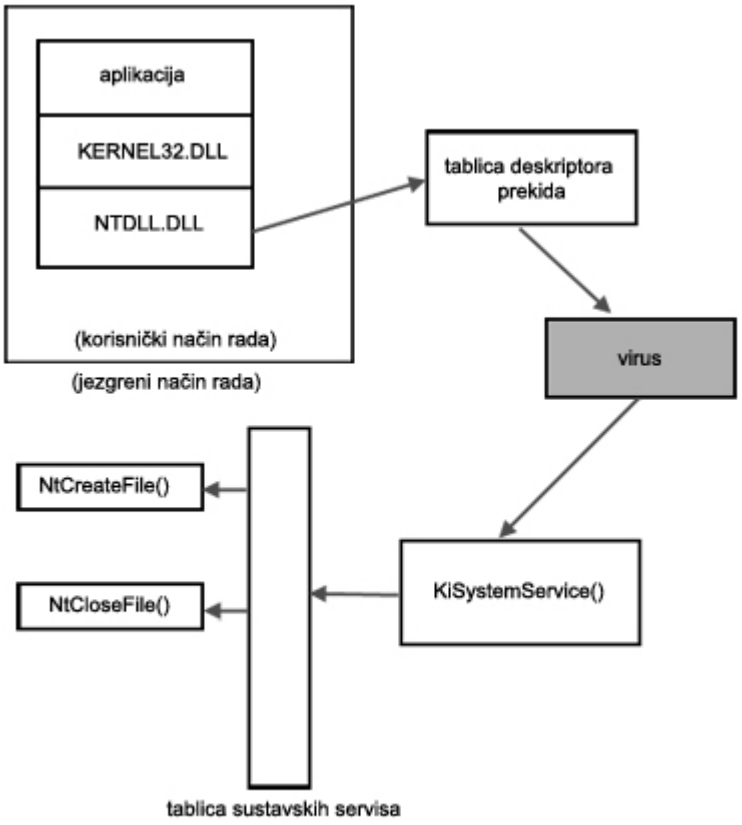
Drugi način je pokretanje virusa prije originalnog programa. Pomoću privremenih datoteka koje sadržavaju samo virus tj. njihovim pozivanjem prijmjerice predavanjem parametara komandnoj liniji, virus postaje zasebni proces.

Koristeći *Service Control Manager-a,* moguće je virus prijaviti kao servis. Servisi su aplikacije koje se pokreću prilikom podizanja *Windows* sustava i rade u privilegiranom načinu rada u pozadini. Od operacijskog sustava one primaju poruke o početku i zaustavljanju [2].

3.4.5 Virusi u jezgrenom načinu rada (Windows NT/2000/XP)

Zapisivanjem odgovarajućeg ključa u *Registry* virus se može prijaviti kao sistemski servis. Kao i ostali *driveri*, potrebno je ponovo pokrenuti sustav, pri čemu će se automatski pokrenuti virus i to u privilegiranom načinu rada. Takav način koristi primjerice virus "*Infis*". Nakon što na navedeni način preuzme kontrolu, *Infis* napada tablicu deskriptora prekida (*interrupt descriptor table*) tako da se zakači na prekid koji poziva *KiSystemService()* funkciju. To mu je omogućeno zbog najviših prava pristupa.

Kada *Win32* aplikacija pokuša pozvati *API* funkciju, to čini preko *Win32* podsustava. Podsustav prevodi dokumentiranu *API* funkciju u nedokumentiranu funkciju sustava. Pri svakom otvaranju neke datoteke na sustavu doći će do prekida koji poziva funkciju *KiSystemService().* Umjesto poziva funkcije aktivirati će se virus. On provjerava ime i ekstenziju datoteke, te provjerava radi li se o *PE* datoteci kako bi ju u tom slučaju inficirao. Ova tehnika je vrlo slična *terminate and stay resident* tehnici napada na *MS DOS* operacijskom sustavu [2].

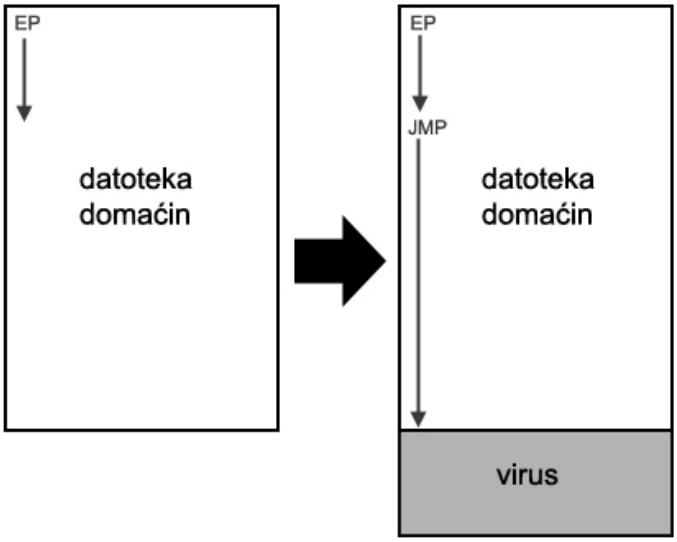


Slika 3.13 Virus u jezgrenom načinu rada

3.5 Tehnike samoobrane zlonamjernih programa

3.5.1 Tehnika prikrivanja pomoću skoka

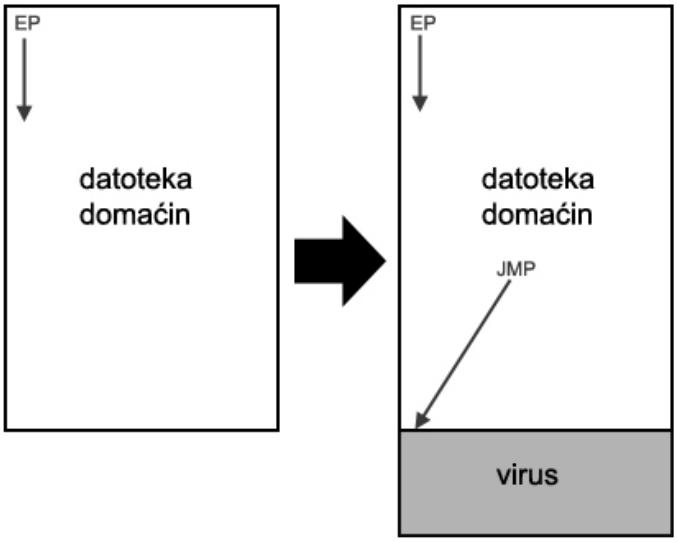
Ova tehnika je zapravo vrlo jednostavna, no svejedno može zavarati antivirusne alate koji koriste heurističke metode traženja *malware-a*. Tijelo virusa se dodaje datoteci domaćinu. Lokacija nije bitna za tehniku skrivanja. Skrivanje se sastoji od jednostavne tehnike da se umjesto mijenjanja ulazne točke na adresu zapisanu u njoj postavlja naredba za skok (*tricky jump obfuscation tehnique*). Skok usmjeruje tok programa na virus. Na taj način izbjegava se mijenjanje ulazne točke što može biti problem za antivirusne programe koji traže tijelo virusa na adresi (ili u blizini) ulazne točke [2].



Slika 3.14 *Tricky jump* tehnika

3.5.2 Prikrivanje ulazne točke virusa

Jedna od najčešćih tehnika kojom se virusi pokušavaju obraniti od antivirusnih programa je i sakrivanje vlastite ulazne točke (*entry point obscuring - EPO*). Pri takvoj infekciji se ne mijenja originalna ulazna točka programa. Umjesto toga se na nasumično (ili teško predvidivo) mjesto u programu postavlja naredba za skok na tijelo virusa. Time postoji mogućnost da tok programa nikad ne dođe do točke da se kontrola predaje virusu. Iako je to nedostatak takve tehnike infekcije, to mu je i prednost jer je antivirusnim programima teže pratiti sve moguće tokove programa kako bi prepoznali virus [2].



Slika 3.15 *Entry point obfuscation*

3.5.3 Nevidljiivi virusi

Ovo je naziv za skupinu tehnika kojom virusi pokušavaju prevariti antivirusne programe te se prema njima učiniti nevidljivima (*stealth viruses*). Općenita tehnika se sastoji od presretanja funkcije ili skupa funkcija koje koriste antivirusni (ali i ostali) programi te mijenjanja povratnih vrijednosti. Na taj način kod traženja virusa antivirusni program dobiva vrijednosti različite od stvarnih što ga onemogućava u prepoznavanju virusa. *Stealth* metoda se pojavila vrlo rano, već kod spomenutog virusa *Brain* koji je presretao prekidni potprogram za pristupanje disku i pri svakom pristupu zaraženom sektoru je prikazivao originalni boot sektor.

Polunevidljivi (*semistealth*) virusi su podskupina *stealth* virusa koji skrivaju samo promjenu veličine inficirane datoteke. Sadržaj datoteke, pa i virus, mogu se vidjeti običnim pristupom datoteci. Koraci koje ta tehnika zahtijeva su:

- Virus postaje rezidentan u memoriji
- Virus presreće funkcije za dohvat datoteka (poput *FindFirstFile(), FindNextFile()*)
- Virus inficira datoteku i označava ju svojom oznakom (*flag*)
- Ukoliko se pristupa već zaraženoj datoteci, vraća originalnu veličinu datoteke u povratnim vrijednostima

Virusi nevidljivi za čitanje (*read stealth*) koriste malo napredniju tehniku. Oni pri pristupu datoteci ili dijelu sustava simuliraju originalan sadržaj. Može se raditi o boot sektoru (kao kod virusa *Brain*). Presretanjem prekidnog potprograma za pristupanje disku virus prikazuje originalni boot sektor umjesto inficiranog boot sektora.

Osim toga, na *MS DOS* operacijskom sustavu postoje virusi koji presretanjem potprograma za čitanje i pomicanje po datotekama vraćaju simulirani originalni sadržaj datoteke. Vrlo je jednostavno za *prepending* viruse da u slučaju čitanja datoteke preskoče tijelo virusa i vrate samo originalnu datoteku.

I na novijim operacijskim sustavima (*Windows*) postoje *read stealth* virusi. Antivirusni programi najčešće koriste standardne C funkcije za otvaranje i čitanje datoteka zbog portabilnosti programa. Presretanjem tih funkcija virus antivirusnom programu prezentira neinficiranu datoteku.

Visoko sofisticirana tehnika za postizanje nevidljivosti je *stealth* na hardverskoj razini. Nju koristi primjerice virus *Strange na* XT sustavima. *Strange* se zakači na *INT 13* prekid. Taj prekid generira hardver kod pristupanja disku kada se napuni međuspremnik (*buffer*). Virus pomoću pristupa (*port*) 6 provjerava sadržaj međuspremnika. Ukoliko je sadržaj inficiran, on prepisuje sadržaj međuspremnika sadržajem originalne datoteke [2].

3.5.4 Kriptirani virusi

Enkripcija (*encrypted viruses*) je jedna od najčešćih tehnika sakrivanja koda virusa, tj. funkcionalnosti virusa, od antivirusnih programa, koji se u najvećem dijelu oslanjaju na pretraživače pomoću potpisa tj. skenere (*scanner*). Enkripcijom tijelo virusa dobiva drugi oblik, neprepoznatljiv za skenere. Taj novi oblik u većini slučajeva nema smisla kao binarni izvršni kod.

Kriptirani virusi sastoje se od dva bitna dijela, konstantnog dekriptora i kriptiranog tijela virusa. Dekriptor je osnova ove metode skrivanja. To je dio koda koji pretvara kriptirano tijelo virusa iz kriptiranog u jasan oblik (i obrnuto). Prije preuzimanja kontrole tijelo virusa i njegova destruktivna funkcionalnost skriveni su enkripcijom.

Algoritam enkripcije najčešće nije složen. Dovoljna je logička funkcija *XOR* kako bi dekriptor jednostavno kriptirao tijelo virusa odabranim ključem. Iako se radi o primitivnom algoritmu enkripcije, to je sasvim dovoljno da zavara neke antivirusne programe.

Faktor koji jako utječe na težinu prepoznavanja takvih virusa je odabir tj. promjena ključa enkripcije. Ukoliko bi ključ enkripcije bio uvjek isti, antivirusni *scanner-i* bi mogli tražiti i kriptirano tijelo virusa, bez obzira što ono u takvom obliku nema smisleni i destruktivan oblik. Primjer virusa *Cascade* pokazuje jednostavnu metodu promjene ključa. Virus svoj kriptirani kod postavlja na kraj inficirane datoteke, i kao ključ enkripcije koristi udaljenost svog koda od početka datoteke. Ključ enkripcije će prema tome biti jednak samo u slučaju da se radi o dvije jednako velike datoteke domaćina.

Najčešći način pronalaženja takvih virusa je traženje statičkog dekriptora koji je jedini nekriptirani dio virusa. Problem kod takvog pristupa je što antivirusni alat katkada ne može točno odrediti inačicu virusa, što može biti bitna informacija potrebna za dezinfekciju datoteke. Osim toga, moguće je dobivanje lažno pozitivnih rezultata traženja virusa kod programa koji sadržavaju takve dekriptore, primjerice za *antidebugging.*

Postoji još velik broj tehnika kako autori kriptiranih virusa dodatno otežavaju njihovu detekciju. Jednostavna metoda je korištenje raznih smjerova kretanja petlje za dekripciju, uz mijenjanje vrijednosti ključa ovisno o broju iteracije. Neki virusi koriste razne metode nelinearne dekripcije, pa čak i jake enkripcijske algoritme poput *IDEA* algoritma. Virus mora na neki način sadržavati i ključ enkripcije, pa se stoga ne može smatrati jako sigurnim, ali svejedno otežava zadatak antivirusnim alatima koji moraju implementirati algoritam dekripcije.

Neki virusi ne čuvaju ključ enkripcije. Umjesto toga se koriste iscrpnim pretraživanjem (*brute force*) kako bi dekriptirali tijelo virusa. Ta metoda se još naziva nasumični algoritam dekripcije (*random decryption algorithm – RDA*) [2].

3.5.5 Oligomorfni virusi

Prvi korak autora virusa nakon stvaranja kriptiranog virusa je zaštita dekriptora od detekcije. Oligomorfni virusi (*oligomorphic viruses*) sadrže dekriptore koji na razne načine mijenjaju oblik kroz generacije virusa.

Najjednostavnija metoda je korištenje većeg skupa dekriptora od kojih će se samo jedan koristiti u svakoj generaciji virusa. Poznati su virusi koji u sebi sadržavaju više desetaka dekriptora, od kojih se aktivni dekriptor odabire na slučajan način.

U oligomorfne viruse se ubrajaju i virusi koji mogu blago mutirati kod dekriptora [2] [15].

3.5.6 Polimorfni virusi

Polimorfni virusi (*polymorphic viruses*) su slijedeća razina izbjegavanja *scanner-a*. Osnovni primjer polimorfnog virusa može se vidjeti na prvom takvom primjerku koji se pojavio *in the wild,* virusu „*1260*". Taj virus ima sposobnost umetanja viška koda (*junk code*) u kod dekriptora. Taj umetnuti kod nema nikakvu funkciju, nego služi samo za zavaravanje *scanner-a*. Količina umetnutog koda nije konstantna, pa i veličina dekriptora varira. Osim toga, dekriptor se sastoji od tri skupine instrukcija (*prolog,decryption,increments*) od koje se mogu permutirati tako da kostur dekriptora mijenja svoj oblik. Na taj način, i dekriptor može poprimiti velik broj funkcionalnih oblika koje je teško prepoznati traženjem specifičnog uzorka.

Slijedeći korak u razvoju ove tehnike zaštite virusa od antivirusnih programa bio je stvaranje mutatorskog stroja (*mutation engine*) koji je baziran na modularnom razvoju, zvan *The Dark Avenger Mutation Engine (MtE)*. Taj *engine* se dodaje virusu, te se poziva kao funkcija sa parametrima predanim kroz određene registre. *MtE* tada stvara polimorfnu zaštitu na postojeći virus. To je olakšalo posao autorima virusa koji su početnici.

Danas je poznat velik broj mutatora. Prednost postojanja ovakvih *engine-a* za borbu protiv virusa je u tome što jednom kada se pronađe metoda za detekciju mutiranog dekriptora, ona vrijedi za širok skup virusa koji koriste taj određeni mutator. No nažalost, otkrivanje metode detekcije je često teško i specifično za svaki pojedini mutator [2] [15].

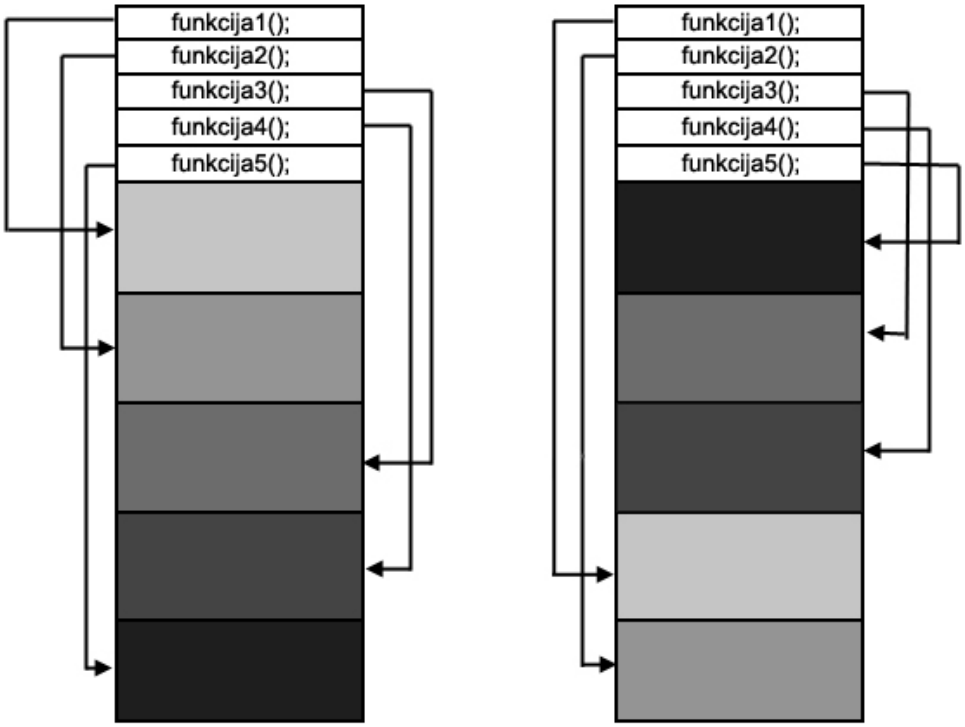
3.5.7 Metamorfni virusi

Pisanje polimorfnih virusa (*metamorphic viruses*) je težak posao, i takvi virusi često imaju velik broj grešaka koje ih čine neuspješnima. Osim toga broj uspješnih vanjskih mutatora nije velik, a za mnoge već postoje obrambene tehnike. Zato autori

virusa koriste drugačiju metodu zvanu metamorfizam, koja u biti predstavlja polimorfizam tijela virusa. Takvi virusi ne koriste enkripciju kako bi mijenjali svoj oblik iz generacije u generaciju već pokušavaju promijeniti svoj oblik, a zadržati istu funkcionalnost.

Jednostavni metamorfizam postiže se korištenjem drugačijeg skupa registara u instrukcijama virusa. Moguće je postići da virus radi i postiže jednak učinak korištenjem drugih registara (na taj način da jedna instrukcija ne utječe negativno na registar druge). Kod virusa se time mijenja, a time nastaje nemogućnost *scannera* da pomoću jednog uzorka za prepoznavanje (niza instrukcija) prepozna svaku iduću inačicu virusa. Prepoznavanje se može poboljšati korištenjem uzoraka sa zamjenskim znakovima (*wildcards*). Ta metoda svejedno nije nužno uspješna za sve viruse koji rade na taj način. Postoji mogućnost da dijelovi uzorka virusa koji su konstantni nisu reprezentativni samo za virus nego i za obične programe.

Još jedan jednostavan primjer je korištenje niza kratkih potprograma ili funkcija, koje same za sebe ne predstavljaju uzorak koji se može koristiti za prepoznavanje virusa. Virus poziva te potprograme istim redoslijedom, a jedina razlika je razmještaj potprograma. Zbog promjene u razmještaju ne može se izdvojiti jedan kontinuirani niz instrukcija koji sigurno predstavlja taj virus. Ako je broj potprograma jednak ***n*** onda se broj različitih inačica virusa može jednostavno izračunati kao ***n!***.



Slika 3.16 Metamorfni virus

Polimorfni virusi u nekom trenutku obično pune svoj dekriptirani kod u memoriju kako bi se izvršili. U tom trenutku moguće je prepoznati virus pretraživanjem pomoću potpisa. Metamofni virusi koriste tehnike kojima mijenjaju svoj kod na takav način da ni bez enkripcije ne mogu biti prepoznati pomoću pretraživača.

Jedna od takvih tehnika je umetanje i brisanje instrukcija za skok (*JMP*) u svoj kod. Te instrukcije za skok jednostavno pokazuju na iduću instrukciju. Funkcionalnost virusa se ni malo ne mijenja. Razlika može biti u vremenu izvođenja, ali tu je posljedicu teško iskoristiti za detekciju. U svakoj novoj generaciji, virus ima *JMP* instrukcije na drugim mjestima, i praktički je nemoguće izdvojiti uzorak koji *scanner* može koristiti za detekciju.

Metamorfni virusi mogu sadržavati tablice i sustave za zamjenu istoznačnih instrukcija. Neki učinci koji nastaju nekom instrukcijom (promjena sadržaja registra i slično) mogu se postići i drugim instrukcijama ili skupovima instrukcija (npr. *sub eax,eax i xor eax,eax*).

Korištenjem instrukcija za skok ili permutiranjem nezavisnih instrukcija (ne utječu jedna na drugu) moguće je permutirati instrukcije iz koda virusa na taj način da će virus u svakoj generaciji izgledati drugačije, no tok programa će biti logički isti.

Metode koje stvaraju nateže prepoznatiljive viruse su integracija virusa sa kodom i mutacija domaćina. Neki virusi svoj kod umeću u dijelićima unutar koda domaćina. Za to je potrebno ažurirati i instrukcije domaćina kako bi program i dalje radio. Mutacija domaćina označava permutiranje dijelova domaćina i zamjena njegovih instrukcija ekvivalentnima. Mutacijom datoteke domaćina moguće je zahvatiti i neki prethodno inficirani virus i time stvoriti novu mutaciju i drugog virusa. Navedene metode vrlo su složene [2] [15].

3.5.8 Retrovirusi

Retrovirusi su virusi koji specifično napadaju antivirusne programe, programe za vatrozid i ostale sigurnosne programe kako bi im zaobišli djelovanje ili ga onemogućili.

Činjenica je da većina korisnika *Windows* operacijskog sustava koristi korisničke račune s administratorskim ovlastima. Te ovlasti među ostalim omogućavaju gašenje procesa i uništavanje datoteka koje pripadaju sigurnosnim programima. Virusi pokrenti u tom načinu rada također će imati takve ovlasti.

Napad na antivirusni program većini korisnika ne predstavlja velik problem, ali na taj način retrovirusi otvaraju slobodan put infekciji virusima koji su inače jednostavno zaustavljani.

3.6 Računalni crvi

3.6.1 Općenita svojstva računalnih crva

Računalni crvi su maliciozni programi koji se šire sa sustava na sustav putem mreže. Takav način širenja zahtjeva određene mehanizme koji mu to omogućavaju. Iz tog se razloga računalni crvi mogu raščlaniti na pojedine logičke komponente. Glavni dijelovi računalnog crva su tragač mete (*target locator*), prenositelj infekcije (*infection propagator*), te nekoliko opcionalnih modula poput udaljene kontrole (*remote control*), upravljača životnog ciklusa (*life-cycle manager*) i tereta (*payload*).

3.6.2 Tragač mete

Tragač mete (*target locator*) je jedan od najvažnijih dijelova računalnog crva. Kako bi se crv proširio na novi sustav potrebno je naći adrese dostupnih sustava. Adresa može imati više različitih značenja. Tako se može raditi o *e-mail* adresi, o *IP* adresi sustava na lokalnij mreži itd.

Jedna od najjednostavnijih metoda širenja mrežom je putem *e-mail* poruka. Takvi virusi moraju naći nove *e-mail* adrese kako bi poslali svoju kopiju na novi sustav. Postoji više načina pronalaženja novih adresa. Adresari sustava i *e-mail* programa su početno mjesto potrage. Izvlačenje adresa iz njih ne predstavlja teškoću i crv se pomoću tako sakupljenih adresa može eksponencijalno širiti putem Interneta.

Parsiranje datoteka koje sadrže *e-mail* adrese druga je metoda traženja novih meta. Mnogi crvi traže datoteke sa HTM, PHP, ASP, WAB i ostalim ekstenzijama i parsiraju ih u potrazi za *e-mail* adresama. Parsira se heurističkim metodama, koristeći neke pretpostavke o tome koji dio teksta može biti *e-mail* adresa, a za koji dio to nije vjerojatno. To stvara ovisnost uspješnosti crva o formatu tih datoteka koji nije u svim slučajevima jednak. Iako se čini da je to veliki nedostatak takvog pristupa, praksa pokazuje da postoji velik broj uspješnih crva koji koriste tom metodom.

Metoda pronalaženja *e-mail* žrtava vrlo poznata kod slanja *spam* pošte koristi *NNTP (network news transfer protocol*). Korištenjem *news* klijenta moguće je pretražiti razne poslužitelje i doći do više milijona *e-mail* adresa.

E-mail adrese mogu se sakupljati i putem web stranica, poput pretraživača (*Google* itd.) ili korištenjem stranica sa osobnim podacima korisnika programa za komunikaciju (*ICQ*) i slično.

Na mrežama sa *Windows* operacijskim sustavima, jednostavno je pronaći ostale lokalno spojene čvorove (računala) korištenjem funkcija operacijskog sustava. Nakon što su čvorovi popisani, postoji više tehnika kako ih učiniti metama. Postoji više

sigurnosnih postavki sustava koje mogu olakšati taj postupak ukoliko su pogrešno postavljene. Administratorske ovlasti na domenama *NT* sustava omogućavaju kopiranje, pisanje, pa čak i izvršavanje programa na računalima koja pripadaju toj domeni.

Mrežni resursi bez postavljenih lozinki za pristup na administratorskoj razini očit su cilj za širenje računalnog crva. Slabe lozinke [23] mogu se razbiti korištenjem *dictionary* napada, tj. isprobavanjem često korištenih lozinki.

Čak i jake lozinke [23] mogu se probiti u dovoljno dugom vremenu. To je moguće izvršiti *brute-force* napadom, što se može pospješiti i podjelom posla između više instanci crva u mrežnom okruženju. Osim toga, moguće je standardnim metodama pratiti promet na mreži te hvatati administratorske podatke, uključujući korisničko ime i sažetak (*hash*) lozinke. Nakon hvatanja *hash* vrijednosti lozinke, pomoću *dictionary* ili *brute-force* napada otkriva se stvarna lozinka.

Nove mete moguće je pronalaziti pretraživanjem svih *IP* adresa uz izbjegavanje onih koje vjerojatno nisu odgovarajuće. Taj postupak može dugo trajati, zbog velikog broja mogućih odredišta. Neki računalni crvi odabiru nasumične adrese i isprobavaju radi li se o nezaštićenom sustavu. Uz ovu metodu obično se koristi iskorištavanje ranjivosti sustava (*exploit*) [2].

3.6.3 Prenositelj infekcije i izvršavanje koda

Računalni crvi se razlikuju po tehnikama kojima prebacuju maliciozni kod na nove sustave (*infection propagator*) i kako se taj kod aktivira. Neki crvi se oslanjaju na postojanje već instaliranog (malicioznog) *back-door* programa na računalu. Oni provjeravaju odabrane mete šaljući zahtjeve specifične za neki „popularni" *back-door* program (npr. *Back Orifice*). Ukoliko meta odgovori odgovarajućim odgovorom, crv se jednostavno širi na to računalo koristeći funkcionalnost instaliranog *back-door* programa, koja uobičajeno podržava i udaljeno pokretanje programa.

Drugi virusi se često oslanjaju na to da će biti pokrenuti od strane korisnika misleći da se radi o običnom programu. Sve veća popularnost P2P (*peer-to-peer*) programa i programa za slanje poruka (*instant messaging*) čini dijeljene podatke jednostavnom metom za računalne crve. P2P programi služe za razmjenu podataka između udaljenih računala (korisnika). Računalni crvi napadaju datoteke u njihovim dijeljenim direktorijima (*shared folders*), najčešće prepisivanjem originalnog sadržaja vlastitim kodom, ali poznati su i *prepending* i *appending* crvi. Autori takvih crva računaju da će udaljeni korisnici kopirati inficirane datoteke i izvršiti ih na svojim računalima. Na sličan način se koriste programi za *instant messaging*. Računalni crv koristeći taj program šalje drugim korisnicima (odabranim prema vlastitom kriteriju) datoteke ili linkove na datoteke i stranice koje su inficirane. Korisnici kojima je to poslano moraju pokrenuti takve datoteke i time se infekcija širi.

Ipak, daleko najčešća metoda propagacije putem mreže je korištenje *e-mail* poruka. Najčešći razlog širenja takvih crva je nesigurno korištenje računala. Crvi u tim porukama nude sadržaj kojem je svrha prevariti korisnika i natjerati da ga pokrene. Često u kombinaciji sa time dolazi i lažiranje zaglavlja *e-mail* poruka, što postiže dojam da je poruka došla od sigurnog izvora.

Jednostavan način kako crvi šalju zaražene poruke je umetanjem *e-mailova* u „poštanske sandučiće" (*mailbox*) programa za slanje *e-mail* poruka. Dodavanjem u izlazni sandučić crv računa da će *e-mail* klijent sam poslati inficirane poruke novim metama.

Neki računalni crvi pokušavaju dodati svoju kopiju u sve izlazne *e-mail* poruke. To se može izvesti izmjenom datoteke u kojoj je zapis o *e-mail* poslužiteljima. Umjesto originalnog poslužitelja postavlja se lokalna adresa (127.0.0.0, *localhost*). Crv se ponaša kao posrednik (*proxy*) i nakon što svakoj poruci doda svoju kopiju prosljeđuje ju na originalni poslužitelj.

U *Registry-ju Windows* operacijskog sustava postoje zapisi o korištenim *e-mail* poslužiteljima i korisničkim računima. Crvi mogu koristiti *SMTP* protokol i slati *e-mail* poruke. Ukoliko ne postoje takvi zapisi crvi često koriste fiksne adrese poslužitelja, što može uzrokovati zagušenje kod velikog broja zaraženih računala.

Format u kojem se maliciozni kod prenosi preko mreže može biti različit. Poruke mogu biti u *HTML* obliku, i iskorištavati neke ranjivosti *e-mail* klijenata. Može se raditi i samo o linku na zaraženu stranicu. Današnje *e-mail* poruke se standardno šalju u *MIME* formatu, pa su crvi često kodirani u tom obliku.

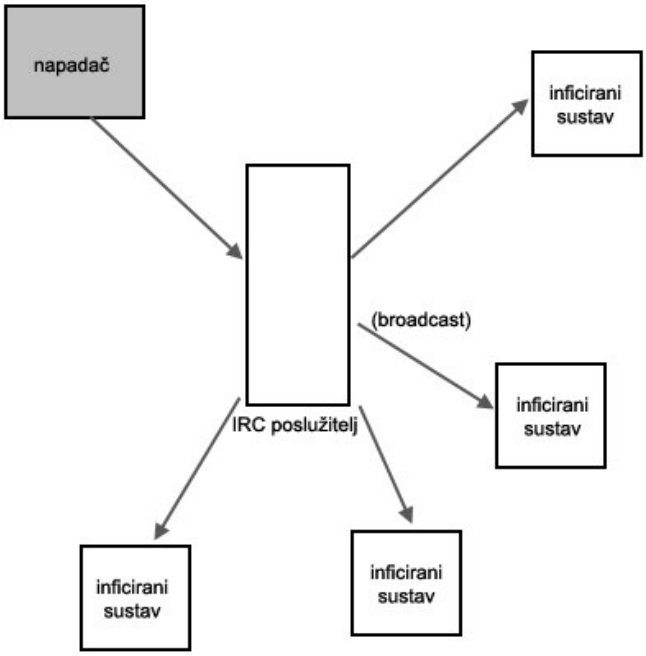
U *Unix/Linux* okruženju sustavi imaju mogu biti napadnuti jednostavnim korištenjim naredbi poput *rsh, rlogin, rcp, rexec* [6], ukoliko nisu postavljene lozinke ili ako se lozinke mogu razbiti *dictionary* napadom. Na sličan način mogu biti napadnuti *Microsoft SQL* poslužitelji, korištenjem funkcije *xp_cmdshell* [5].

Crvi mogu ciljati i ranjivosti poslužiteljskih sustava, koji nedovoljno provjeravaju sadržaj unosa. Za te napade se koristi umetanje koda (*code injection attack*). Crv ranjivom poslužitelju šalje kod koji se izvršava na poslužitelju, vjerojatno u privilegiranom načinu rada. Ovim napadom moguće je pokrenuti dretvu koja sluša određeni TCP/UDP pristup i čeka naredbe komandne linije tj. ljuske (*shell*). Takav napad se obično izvršava prema sustavima sa *Unix/Linux* okruženjima [2].

3.6.4 Ostale komponente računalnih crva

Autori računalnih crva često žele mogućnost upravljanja udaljenim inficiranim sustavima. Česta svrha udaljenog pristupa je iniciranje DDoS (*distributed denial of* service) napada. Crv sadržava modul koja pruža određenu *back-door* funkcionalnost. Crv javlja autoru svoju adresu, i obavještava ga o novo-inficiranom računalu. Primjer za

to je korištenje *IRC* (*Internet Relay Chat*) kanala kao sredstvo za odašiljenje *broadcast* poruka zaraženim računalima.



Slika 3.17 Upravljanje DDos napadom putem *IRC* poslužitelja

Crvi katkad kontroliraju svoj životni vijek. Unaprijed postavljaju period izvršavanja nakon čega prekidaju izvođenje. Naravno, uočeni su i primjerci koji sadržavaju greške zbog kojih se izvršavaju zauvijek.

Neki crvi imaju sposobnost pokretanja novih inačica samih sebe (*update*). Na web adresama (koje kontrolira autor crva) se nalaze nove kopije crva. Crv provjerava ima li novih („boljih”) verzija i ukoliko ima kopira ih i pokreće.

Teret (*payload*) računalnih crva kao i kod običnih virusa može imati širok spektar više ili manje destruktivnih posljedica, poput uništavanja podataka, krađe podataka, zagušenja mreže, DoS i DDoS napada protiv ciljanih sustava [2].

4. Tehnike obrane od malware-a

4.1 Pretraživači prve generacije

4.1.1 Pretraživanje nizova

Usporedba niza *byte-ova*, izvučenih iz koda *malware-a*, sa potencijalno zaraženom datotekom ili dijelom sustava i dalje je jedna od najpopularnijih tehnika koju koriste antivirusni alati. Iako se radi o najjednostavnijoj tehnici, ipak njena je učinkovitost vrlo dobra za određene vrste *malware*.

Iz virusa se vadi niz *byte-ova* koji je reprezentativan za taj virus i ne može se pojaviti u neinficiranim datotekama. Taj niz se naziva „potpis" (*signature*) i pohranjuju se u baze podataka antivirusnih programa. Ti antivirusni programi koriste pretraživače kojii pretražuju određene dijelove datoteka ili sustava uspoređujući ih sa potpisima u bazi (koristeći određene algoritme). Za *DOS* viruse često je dovoljan potpis od samo 16 *byte-ova.* Za 32-bitne viruse, koji su često pisani u višim programskim jezicima potpisi su obično duži.

Teškoća na koju prvenstveno nailazi ovaj pristup je brzina pretraživanja, jer se radi o velikim količinama potpisa virusa. Osim toga neki potpisi mogu biti reprezentativni za više inačica virusa, što može uzrokovati probleme prilikom dezinfekcije datoteke ili dijela sustava.

4.1.2 Zamjenski znakovi

Česti dodatak pretraživačima (skenerima) nizova prve generacije je korištenje zamjenskih znakova (*wildcards*). Umjesto kontinuiranih nizova *byte-ova* dopušta se korištenje zamjenskih znakova unutar potpisa *malware-a*. Zamjenski znak može označavati jedan ili određen broj promjenjivih *byte-ova* unutar potpisa virusa. Neki pretraživači podržavaju i korištenje regularnih izraza [7].
Primjer zamjenskih znakova korištenih u *Clam AV* [22] antivirusnom programu otvorenog koda:

Tablica 4.1 Zamjenski znakovi kod *Clam AV* antivirusnog alata

??	Zamjena za jedan *byte*
*	Zamjena za bilo koliko *byte-ova*
{n}	Zamjena za **n** *byte-ova*
{-n}	Zamjena za **n** ili manje *byte-ova*
{n-}	Zamjena za **n** ili više *byte-ova*
(a|b)	Zamjena za **a** ili **b** (može i više znakova

4.1.3 Ubrzavanje pretraživanja

Za usporedbu nizova potrebno je koristiti brze algoritme za usporedbu poput *Boyer-Moore* algoritma [24]. Osim toga postoje metode korištenja vrijednosti sažetaka potpisa virusa (ili njegovog dijela) koji ubrzavaju postupak pretraživanja. Navedene metode biti će opisane u daljnjem tekstu.

Na brzinu pretraživanja utječe i koji će dio datoteke ili dijela sustava biti pretražen. Često se ne mora pretražiti cijelu datoteku kako bi se došlo do pozitivnog ili negativnog rezultata usporedbe.

Pretraživanje početka i kraja datoteke (*top and tail scanning*) je jedan od takvih pristupa. Antivirusni program pretražuje samo određeno duge dijelove početka i kraja datoteke jer je to mjesto gdje će se nalaziti *prepending* i *appending* virusi. Time se štedi vrijeme koje bi inače bilo utrošeno na pretraživanje unutrašnjosti datoteke. Takav pristup naravno nije prigodan za sve vrste virusa, ali u navedenim slučajevima može dovoljno ubrzati postupak.

Pretraživanje od ulazne točke (*entry point scanning*) i skeniranje od fiksno udaljene točke (*fixed point scanning*) još više ubrzavaju proces pretraživanja. Korištenjem struktura izvršnih datoteka, antivirusni programi mogu otkriti poziciju u datoteci od koje počinje izvršavanje programa. To je ujedno i često mjesto gdje počinje tijelo virusa. Neki antivirusni programi prate tok instrukcija za skok i na taj način pokušavaju upariti znakove potpisa sa sadržajem potencijalne datoteke domaćina, bez pretraživanja nepotrebnih dijelova datoteke. Ukoliko se na početku virusa (a time i na ulaznoj točci) ne nalaze *byte-ovi* pogodni za usporedbu moguće je zadati fiksnu točku udaljenosti od ulazne točke od koje kreće usporedba (*fixed point scanning*).

4.2 Pretraživači druge generacije

4.2.1 Pametno pretraživanje

Antivirusni pretraživači (skeneri) druge generacije koriste pametno pretraživanje (*smart scanning*) tj. pametnu usporedbu sa potpisima virusa. Ta metoda podrazumijeva da virusi mogu ubacivati redundantne (*junk*) instrukcije u svoj kod i time mijenjati vlastitu strukturu. Pametno pretraživanje se odnosi na preskakanje funkcija koje nikako logički ne utječu na kod. Takvo pretraživanje se može izvesti upotrebom konačnih automata [8] [16].

Ovakav pristup prikladan je i za viruse koji dolaze u tekstualnom obliku (makro ili interpretatorski jezici) koji lako mogu mijenjati oblik dodavanjem praznih prostora (*white space*) u kod programa. Vizualni izgled i oblik se pri tome mijenjaju ali logička struktura ostaje apsolutno ista.

4.2.2 Detekcija kostura programa

Ova metoda je posebno prikladna za otkrivanje makro virusa. Ideja se zasniva na tome da se iz koda izbace sve instrukcije (naredbe) koje nisu prijeko potrebne za očuvanje osnovne strukture programa. To uključuje naravno i strukturne elemente poput *white spaces* znakova. Na taj način ostaje samo „kostur" malicioznog programa. Ista metoda se primjeni na provjeravani program (domaćin) i traži se podudarnost.

4.2.3 Približno točna identifikacija

Pretraživači koji se oslanaju približno točnu identifikaciju (*nearly exact identification*) koriste se više dijelova potpisa virusa koji su konstantni. Tako za svaki maliciozni program postoje dva ili više niza (potpisa) koji se traže unutar potencijalne datoteke domaćina. Ako pretraživač pronađe samo jedan od tih nizova unutar datoteke, moguće je da se radi o inačici tog virusa, ali sigurna identifikacija nije moguća. To može utjecati na mogućnost dezinfekcije, jer uklanjanje virusa iz inficirane datoteke obično zahtijeva da je poznata točna struktura, veličina i pozicija virusa.

Ova metoda se često koristi u kombinaciji sa korištenjem kontrolnih suma (*checksum*) (kriptografskih, hash, CRC32…) računatim nad određenim statičkim dijelovima virusa. Na taj način se štedi prostor potreban za pohranu baze potpisa virusa. Za približno točnu identifikaciju koristi se više kontrolnih suma izračunatih nad dijelovima virusa.

4.2.4 Točna identifikacija (*exact identification*)

Radi se o vrlo sličnom postupku kao kod približno točne identifikacije, ali se za izračunavanje kontrolne sume ne koriste samo određeni dijelovi virusa nego svi njegovi konstantni *byte-ovi.* Promijenjivi podaci se izbacuju kako bi se stvorila mapa konstantnih *byte-ova*. Konstantni podaci (npr. poruke autora virusa) se mogu koristiti za izračunavanje kontrolne sume, ali njihova promjena (koja ne uzrokuje bitno drugačije ponašanje virusa) može poremetiti kontrolnu sumu.

4.3 Algoritamske metode pretraživanja

4.3.1 Značenje algoritamskih metoda

Prvi antivirusni programi sadržavali su implementacije nekih od algoritama (jednostavnije ili složenije) za traženje potpisa virusa. Razvojem složenosti modernih virusa postajalo je sve manje moguće detektirati i uklanjati viruse pomoću općenitih

algoritama. Pomalo su dodavani moduli koji su bili specifični za rukovanje pojedinim virusom koji nije mogao biti otkriven bez korištenja nekih specifičnosti.

Nadogradnje antivirusnom softveru moraju se izdavati brzo nakon pojave virusa. Takva brzina programiranja kao posljedicu može ostaviti pogreške koje su ustanju srušiti antivirusni softver.

Radi jednostavnosti i portabilnosti antivirusni programi razvijaju vlastite programske jezike i interpretere koji služe razvijanju zasebnih algoritama za svaki pojedini virus koji se ne može obuhvatiti općenitom metodom. Uglavnom se radi o specijaliziranim jezicima koji podžavaju pristup datotekama ili dijelovima sustavima i funkcijama za pretraživanje.

Takvi jezici i virtualni strojevi koji ih interpretiraju omogućavaju da „algoritamski kod" bude portabilan na različite platforme bez promjena, jer ovise samo o antivirusnom alatu, a ne o operacijskom sustavu ili računalnoj arhitekturi. Time se povećava brzina izdavanja takvih algoritamskih nadogradnji nakon pojave novih malicioznih programa.

Algoritamske metode mogu biti vrlo vremenski zahtjevne. Često je potrebno izvršiti velik broj iteracija petlji koje traže pojedini virus. I ostale jednostavnije metode traženja virusa mogu trajati vrlo dugo uzmemo li u obzir broj potpisa koje sadržavaju baze antivirusnih programa. Zbog toga potrebno je implementirati određenu vrstu filtriranja, tj. određivanja koje će se metode koristiti i koji će se *malware* prokušati detektirati. Neki virusi napadaju samo određene dijelove sustava, poput određenih datoteka. Tako primjerice nije potrebno vršiti provjeru nad potencijalnom datotekom domaćinom za virus koji napada isključivo boot sektore [2].

4.3.2 Detekcija dekriptora

Za točnu detekciju kriptiranih virusa potrebno je koristiti algoritamske metode. U inficiranoj datoteci traži se statički deskriptor prema poznatom potpisu. Razni virusi mogu biti sakriveni korištenjem istog dekriptora. Algoritam za detekciju mora koristeći tijelo dekriptora pretvoriti tijelo virusa. Kada je tijelo virusa dekriptirano, može se točno odrediti o kojem se virusu radi, koristeći neku od prije navedenih metoda.

Polimorfni virusi mutiraju dekriptore kako bi otežali proces detekcije. No svejedno, obično bar jedan *byte* dekriptora ostaje konstantan, čak i kod najjačih mutacijskih *engine-a*. To se može iskoristiti kao početna točka za algoritamsku detekciju dekriptora.

Postoji i drugačiji pristup kriptiranim virusima od detekcije dekriptora, nazivan *X-ray* (rendgenska zraka) pretraživanje koja označava napad protiv enkripcije virusa. Proučavanjem metoda enkripcije moguć je kriptoanalitički napad [9] na dijelove datoteka za koje se predviđa da bi mogli sadržavati tijelo virusa. Takva metoda čak i kod

jednostavnih enkripcija može biti vremenski dosta zahtjevna, ali je vrlo općenita što ju čini korisnim oruđem [2].

4.4 Emulacija koda

Ovo je tehnika detekcije *malware-a* koja se oslanja na izvršavanje potencijalne datoteke domaćina u virtualnom računalu. Virtualno računalo mora podržavati korištenje virtualnih registara koji oponašaju stvarno računalo. Osim toga, dohvat i pisanje po virtualnoj memoriji mora odgovarati stvarnoj memoriji. Specifičnosti operacijskog sustava poput *API* funkcija moraju biti dostupni unutar virtualnog računala. Ovisno o kvaliteti oponašanja stvarnog sustava emulator može bolje ili lošije zavarati virus da se se izvršava na željenoj platformi.

Starije implementacije emulatora su koristile *debuggerske* funkcije procesora kako bi pratile izvršavanje koda. Taj način nije dovoljno siguran jer virusni kod može izaći izvan okvira emulacije. Današnji emulatori se osnivaju na strukturama koje predstavljaju komponente poput registara i memorije, te velike uvjetne naredbe (*switch)* koja za svaku instrukciju koja dođe na red za izvršavanje pokušava virtualno izvršiti akciju kakva bi se dogodila na stvarnom sustavu.

Emulacija koda dobro je oruđe za otkrivanje kriptiranih i polimorfnih virusa. U nekom trenu izvršavanja, polimorfni virusi dekriptiraju tijelo virusa i predaju mu kontrolu. U tom trenu moguće je jednostavnim pretraživanjem pomoći potpisa provjeriti da li se u datoteci nalazi virus. Emulacija omogućuje da se u unaprijed definiranom trenutku obavi takvo pretraživanje. Problem koji se pri tome susreće je odabir pravog trenutka za pretraživanje, s obzirom da bi pretraživanje u pogrešnim trenucima bilo previše vremenski zahtjevno. Kod kriptiranih virusa čiji su dekriptori konstantni, moguće je izvršiti pretraživanje nakon određenog broja izvršenih instrukcija. Polimorfni virusi mijenjaju oblik svojih dekriptora, pa je potrebno odrediti drugačiji „okidač”. Za većinu polimorfnih virusa virtualni trenutak za pretraživanje može se odrediti praćenjem aktivnih instrukcija. Instrukcija se naziva aktivna ako uzrokuje promjenu dvije uzastopne memorijske lokacije. Po tome se može prepoznati standardno ponašanje dekriptora.

Drugi način je korištenja profila za dekriptore polimorfnih virusa. Većina tih dekriptora ima svega nekoliko instrukcija, pa se može zaključiti da kad se izvrši instrukcija koja nije dio profila, da je time započelo izvršavanje dekriptiranog koda virusa.

Postoje još razni načini za određivanje pogodnog trenutka za pretraživanje datoteke unutar virtualnog sustava. Tako je moguće postaviti prekidne točke na pojedine instrukcije ili niz instrukcija za koje se (za određeni polimorfni virus) smatra da označavaju trenutak kad je tijelo virusa dekriptirano [2].

4.5 Heurističke metode

Većina modernih antivirusnih programa sadži implementacije nekih heurističkih metoda za detekciju virusa. Razlog tome je što raznim metodama sakrivanja virusi onemogućavaju pronalaženje raznim vrstama pretraživanja. Prvi takav primjer su metamorfni virusi koji mijenjaju svoj oblik do te mjere da je nemoguće pohraniti sve potpise virusa pomoću kojih bi se identificirao samo jedan takav virus. Zbog toga potrebno je koristiti heurističke metode koje se oslanjaju na specifičnosti infekcije pojedinim virusom kako bi otkrili infekciju. Neki od takvih pokazatelja su specifični za svaki virus, poput oznaka koje virusi ostavljaju kako bi spriječili višestruku infekciju. Takva oznaka unutar datoteke (koje su obično na fiksnim pozicijama) može se iskoristiti kao pokazatelj da je datoteka (možda) inficirana virusom. Često samo jedan takav pokazatelj nije dovoljno pouzdan da bi se datoteka proglasila inficiranom. Korištenjem više takvih pokazatelja moguće je otkriti neke viruse.

Nedostatak takvog pristupa može biti što veće skupine virusa mogu infekcijom prouzrokovati isti skup pokazatelja. Time se otežava dezinfekcija inficirane datoteke, ali bar se virus može detektirati.

Noviji formati izvršnih datoteka imaju definiranu strukturu, i zaglavlja u kojima su zapisani podaci o strukturi datoteke. Virusi često inficiranjem datoteka ne mijenjaju vrijednosti zaglavlja pa dolazi do nepodudaranja između zaglavlja i stvarnih veličina. Razlog tome da autori virusa katkada ne brinu o sakrivanju prisutnosti virusa na taj način je prvenstveno u tome što će se datoteka (najčešće) uredno izvršiti makar neke veličine u zaglavljima ne odgovaraju stvarnosti [18].

Osim toga, računanje točnih vrijednosti koje bi trebalo obnoviti unutar zaglavlja katkad nije jednostavno. Uzimajući u obzir da metamorfni virusi mijenjaju svoj oblik i veličinu, otkrivanje i vlastite veličine virusu može predstavljati problem. Korištenje raznih metoda sakrivanja i samoobrane koje podrazumijevaju i korištenje pseudoslučajno izabrane pozicije u datoteci domaćinu dodatno otežavaju autorima virusa da na točan način obnove polja zaglavlja.

PE format je prikladan za proučavanje pokazatelja koji se koriste za heurističku detekciju virusa. Proučavanjem stvarnih korištenih primjera moguće je vidjeti o kakvim se pokazateljima radi [2].

Ulazna točka je u zadnjoj sekciji

PE format izvršnih datoteka ima jasno određenu strukturu. Uobičajenim kompiliranjem sav izvršni kod se grupira unutar jedne sekcije (*.text, CODE...*). Praktično, moguće je imati više takvih sekcija, pa čak i da se jedna takva nalazi na kraju datoteke. No, puno je uobičajenije da je ulazna točka programa u zadnjoj sekciji produkt infekcije *appending* virusom. Takav pokazatelj može se iskoristiti za dokaz „sumnje” [2].

Moguća infekcija zaglavlja

Slično kao prethodno navedeni pokazatelj, ulazna točka nakon infekcije može pokazivati na adresu koja nije unutar nijedne sekcije, nego pokazuje na područje prije sekcija, tj. na zaglavlje. Ovakva ulazna točka ne može se stvoriti nekim specifičnim kompiliranjem i vrlo je uvjerljiv pokazatelj da je datoteka inficirana infektorom zaglavlja [2].

Sumnjive karakteristike sekcije

Sve sekcije *PE* datoteke imaju karakteristike zabilježene u zaglavlju datoteke. Među ostalima, to se odnosi na svojstvo da je sekcija izvršna, i može li se po njoj pisati (*writeable*). Ako sekcija ima istodobno oba svojstva, to se može smatrati sumnjivim. Jednako je čudna kombinacija sekcija po kojoj se može pisati, a ne može iz nje čitati. Druge kombinacije poput sekcije koja preuzima kontrolu izvršavanja programa, a ima samo *writeable* svojstvo ili sekcije sa karakteristikama postavljenim na nulu (u obliku zastavica u zaglavlju) također se mogu iskoristiti kao pokazatelji sumnjivih (moguće inficiranih) datoteka [2].

Neodgovarajuća veličina datoteke u zaglavlju

Veličina datoteke zapisana u *SizeOfImage* polju zaglavlja zaokružuje se (na veću vrijednost) kao višekratnik vrijednosti *SectionAlignment.* Velik dio virusa predviđen za *Windows 95* platformu ne zaokružuje tu veličinu točno. Na navedenoj platformi to ne predstavlja problem i takve se datoteke izvršavaju. Na *Windows NT* program punioc (*loader*) odbija pokrenuti takve datoteke [2].

Neodgovarajuća veličina koda u zaglavlju

Neki virusi dodaju izvršnu sekciju u *PE* datoteke. Pri tome neki ne mijenjaju vrijednost *SizeOfCode* polja u zaglavlju. To polje prikazuje ukupnu veličinu sekcija sa izvršnim kodom u datoteci. Ukoliko ta veličina ne odgovara stvarnoj veličini, to je pokazatelj da je datoteci možda dodana sekcija koja sadržava virus [2].

Sumnjiva redirekcija koda

Prepoznavanje virusa koji se koriste *tricky jump* metodom može se također koristiti heurističkim pokazateljima. Binarni kod datoteke se pretvara u asemblerske naredbe i na adresi ulazne točke (ili u njenoj blizini) se traže naredbe za skok. Skokovi u programu nisu uobičajeni na početku programa, i pojava takve naredbe može se smatrati sumnjivom [2].

Višestruka zaglavlja

Virusi koji dodaju vlastito *PE* zaglavlje u inficiranu datoteku također se mogu otkriti korištenjem heurističkih metoda. Polje zaglavlja *lfanew* u starom *DOS* zaglavlju označava adresu *PE* zaglavlja. Ako to polje pokazuje na adresu u drugoj polovici

datoteke i u blizini početka se može naći još jedno *PE* zaglavlje, to se može iskoristiti kao heuristički pokazatelj [2].

4.6 Spriječavanje infekcije usađivanjem (cijepljenje)

Usađivanje ili cijepljenje (*inoculation*) je metoda obrane od virusa kojom se pokušava spriječiti infekciju prije nego što se ona dogodi, za razliku od prepoznavanja već inficirane datoteke. Osnova za takvu metodu je korištenje oznaka koje virusi ostavljaju u inficiranim datotekama kako bi izbjegli višestruku infekciju. Softveri za usađivanje umeću te oznake u neinficirane datoteke. Virus provjerom svoje oznake odustaje od infekcije misleći da je datoteka već inficirana.

Ova metoda bila je prikladna u doba kada je postojao daleko manji broj virusa nego danas. U novije doba, količina postojećih virusa onemogućuje stvarno korištenje takve metode obrane.

Osim prevelikog broja oznaka koje bi bilo potrebno unjeti u svaku potencijalno ranjivu datoteku, metoda ima još neke nedostatke koji ju čine manje učinkovitom. Virusi često koriste neiskorištena ili manje bitna polja zaglavlja izvršnih datoteka kako bi pohranili svoju oznaku. Iz praktičnih razloga i iz ograničenog broja takvih polja neki virusi koriste ista polja, tj. iste pozicije u datotekama. Iz tog razloga, nemoguće je istovremeno cijepiti datoteku protiv svih virusa koji koriste ista polja.

Korištenje velikog broja oznaka u datoteci mijenja veličinu i strukturu datoteke. To može otežati detekciju i dezinfekciju datoteke od virusa koji ne ostavlja nikakvu oznaku.

Osim toga, cijepljenjem se isključuje korištenje navedenih oznaka kao heurističkih pokazatelja da je datoteka inficirana. Uz pretpostavku da su sve datoteke na računalu neinficirane i cijepljene protiv virusa **X**, ako se na računalo kopira datoteka koja je zaražena virusom **X** (što sigurno znači da ima i odgovarajuću oznaku), infekcija se ne može otkriti pomoću te oznake [2].

4.7 Provjera integriteta

Zahtjevnost identifikacije poznatih virusa (*malware-a*) upućuje na potrebu za korištenjem općenitih, generičkih metoda za zaštitu od infekcije. Jedna od tih metoda je provjera integriteta potencijalno ranjivih datoteka. programi za provjeru integriteta koriste kontrolne sume (SHA1, MD5, CRC32...)[9] izračunate nekim poznatim algoritmom, koje se nalaze u bazi podataka. Ta baza se može nalaziti na sustavu, ili na sigurnom izvoru na mreži (Internetu).

Pri svakom pokretanju datoteka na sustavu zaštićenim takvim softverom, provjerava se odgovara li stvarna kontrolna suma onoj zapisanoj u bazi. Ukoliko se dvije navedene sume razlikuju, postoji šansa da je datoteka inficirana. Osim statističke

mogućnosti da inficirana i čista datoteka daju jednake kontrolne sume, postoji još niz nedostataka ove metode.

Glavni nedostatak ove metode je generiranje previše pogrešno pozitivnih rezultata. Velik broj programa sami mijenjaju vlastitu strukturu. To je način da primjerice prilikom instalacije pohrane podatke o konfiguraciji. Osim toga, velik broj programa koristi sustav kojim se sa Interneta skidaju razne zakrpe i integriraju sa instaliranim programom. U velikom broju slučajeva (npr. *Windows Update*) nije očito koje datoteke su izmijenjene. Zbog toga propada mogućnost provjere integriteta pomoću starih vrijednosti.

Antivirusni programi za provjeru integriteta moraju pretpostaviti da je početno stanje u trenutku kada će biti izgrađena baza kontrolnih suma, sigurno tj. da sustav nije inficiran. Ukoliko to nije tako, ovaj način obrane propada: I virusi koriste razne napade kako bi onemogućili provjere integriteta. *Stealth* virusi su jedni od njih, Pri pokretanju inficirane datoteke oni „poslužuju" programu za provjeru integriteta originalnu datoteku domaćina. Antivirus nema razloga izdati pozitivan rezultat, i zaraza se može širiti dalje.

Postoje objekti koji mogu sadržavati viruse, ali koji se i bez njih učestalo mijenjaju, djelovanjem korisnika. Programi u razvoju koji se često kompiliraju, ili *Word* dokumenti koji mogu sadržavati makro viruse samo su neki od takvih datoteka.

Unatoč ovim nedostacima, može se očekivati da će se provjera integriteta sve više koristiti u budućnosti. Takav softver je gledano iz kuta korisnika spor. No uz pojavu sve više virusa koji skrivaju svoju lokaciju (npr. *EPO*) i antivirusni programi koji koriste neku drugu metodu pretraživanje će zbog većeg broja *I/O* operacija postati sve sporiji. Uz pretpostavku da će sve više stvorenog softvera na neki način biti digitalno potpisano pri izdavanju, za zaključiti je da će se smanjiti i količina softvera koja sama sebe mijenja.

Na kraju, najveći problem mogu stvoriti korisnici koji instaliravaju novi softver na računalu. Taj softver može dolaziti iz nesigurnih izvora. Riješenje tog problema leži u drugoj domeni, koja se tiče sigurnosnih politika i dozvola za korisnike [2].

4.8 Zaustavljanje malicioznog ponašanja

Behavior blocker-i su sustavi za zaštitu od malicioznih programa koji ne pokušavaju identificirati virusni kod, nego prate ponašanje programa i pokušavaju zaustaviti ponašanje koje se može smatrati opasnim (*behavior blocking*). Jednostavan primjer je ako jedan program otvori drugu izvršnu datoteku za pisanje. Ovaj pristup je također ograničen jer postoji beskonačan broj mogućih ponašanja koji vode do infekcije dijelova sustava.

Ovakav sustav zaštite je teže za implementirati od prije navedenih sustava za detekciju *malware-a*. Njegova uspješnost ovisi o razini sigurnosti memorije koju održava operacijski sustav. No i protiv *behavior blocker-a* autori virusa stvaraju napade kako bi ih

onemogućili. Virusi mogu postići privilegirane razine izvršavanja (*kernel mode*) što ih čini ravnopravnima sa softverom koji prati i zaustavlja maliciozno ponašanje programa.

Što se tiče implementacije takvih sustava, oni se koriste metodama koje su zapravo slične metodama koje koriste i virusi. *Behavior blocker-i* se zakače za tablice *API* funkcija ili prekidnih vektora, i hvataju njihove pozive. U trenutku poziva provjeravaju kontekst njihovog izvršavanja i prema tome obavještavaju korisnika o sumnjivom ponašanju [2].

5. Praktični rad

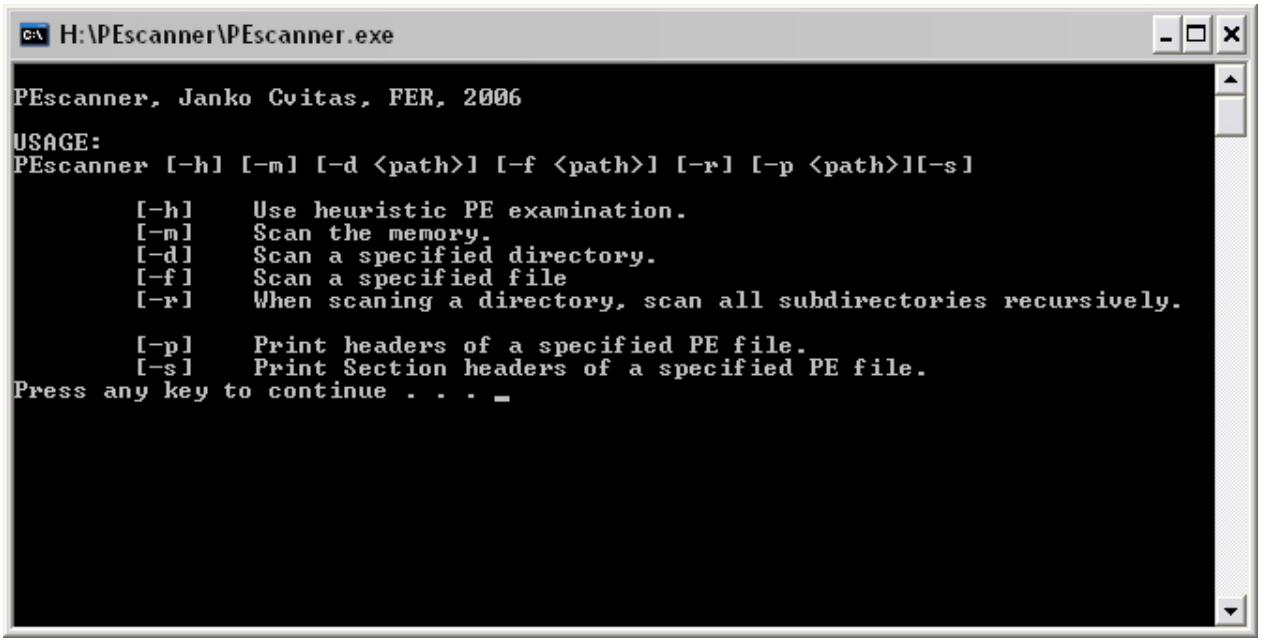
5.1 Uvod

Praktični dio ovog rada predstavlja jednostavan pretraživač koji je sposoban otkriti neke vrste zlonamjernih programa, nazvan *PEscanner.* Kao platforma odabran je *Windows* operacijski sustav. Program se oslanja na specifičnosti operacijskog sustava (*API*...) i nije moguće izravno ga kompilirati za druge operacijske sustave (npr. *Linux*). Objekti pretraživanja prvenstveno su binarne izvršne datoteke no funkcionalnost programa se može djelomično iskoristiti i za druge datoteke. Program je napisan u programskom jeziku C/C++.

5.2 Funkcionalnost programa

5.2.1 Izgled programa

Program je izveden tako da se pokreće iz komandne linije. Izvođenje podfunkcija koje program omogućava pokreće se dodavanjem određenih parametara pri pokretanju. Pokretanje bez parametara ispisuje upute za korištenje.



Slika 5.1 Ispis uputa programa

Tablica 5.1 Parametri za pokretanje programa

-h	Pretraživanje zadanih objekata pomoću heurističkih metoda implementiranih u pretraživaču
-m	Pretraživanje procesa u memoriji.
-d <path>	Pretraživanje određenog direktorija. Nakon parametra mora slijediti putanja do tog direktorija.
-f <path>	Pretraživanje samo jedne datoteke. Nakon parametra mora slijediti putanja do te datoteke.
-r	Rekurzivno pretraživanje . U slučaju da se odabere opcija pretraživanja direktorija, ovaj parametar označava da se pretraživanje mora izvršiti i u svim poddirektorijima (rekurzivno)
-p <path>	Ispis *PE* zaglavlja datoteke. Nakon parametra mora slijediti putanja do te datoteke.
-s	Ispis zaglavlja sekcija *PE* datoteke. Ovaj parametar radi samo ako se navede i –p parametar.

5.2.2 Ispis zaglavlja

```
IMAGE DOS HEADER

Magic number(e_magic): 5a4d
Bytes on the last page of file(e_cblp): 90
Pages in file(e_cp): 3
Relocations(e_crlc): 0
Size of header in paragraphs(e_cparhdr): 4
Minimum extra paragraphs needed(e_minalloc): 0
Maximum extra paragraphs needed(e_maxalloc): ffff
Initial (relative) SS value(e_ss): 0
Initial SS value(e_sp): b8
Checksum(e_csum): 0
Initial IP value(e_ip): 0
Initial (relative) CS value(e_cs): 0
File address of relocation table(e_lfarlc): 40
Overlay number(e_ovno): 0
Reserved words(e_ovno): 0 0 0 0
OEM identifier(e_oemid): 0
OEM information(e_oeminfo): 0
Reserved words(e_ovno): 0 0 0 0 0 0 0 0 0 0
Offset of the new exe header(e_lfanew): e0
```

Slika 5.2 Ispis *DOS* zaglavlja

Ovaj dio programa nije standardna opcija uobičajenih antivirusnih rješenja. Struktura *PE* datoteka može se iskoristiti na mnogobrojne načine kako bi se ustanovilo je li datoteka inficirana virusom. To je osnova za proučavanje virusa na *Windows* operacijskim sustavima. Korištenjem parametra „**–r**“ pri pokretanju programa u komandnoj liniji dobit ćemo ispis svih polja u zaglavlju izvršne datoteke (*PE*). To uključuje staro *DOS* zaglavlje i *PE* zaglavlje koje je bitnije za razumijevanje strukture datoteke.

Prvi dio ispisa prikazuje *DOS* zaglavlje. U zagradama se nalaze nazivi polja kako su definirani u *winnt.h* datoteci koja sadržava sve strukture vezane uz *PE* datoteke. Opisi polja su jednaki komentarima zapisanim u *winnt.h* datoteci.

```
IMAGE FILE HEADER

Signature: 4550

IMAGE NT HEADER

Machine: 14c
Number of sections: 3
Time date stamp: 41107cc3
Pointer to symbol table: 0
Number of symbols: 0
Size of optional header: e0
Characteristics: 10f

IMAGE OPTIONAL HEADER

Magic number: 10b
Major linker version: 7
Minor linker version: a
Size of code: 7800
Size of initialized data: a600
Size of uninitialized data: 0
Address of entry point: 739d
Base of code: 1000
Base of data: 9000
Image base: 1000000
Section alignment: 1000
File alignment: 200
Major operating system version: 5
Minor operating system version: 1
Major image version: 5
Minor image version: 1
Major subsystem version: 4
Minor subsystem version: 0
Size of image: 14000
Size of headers: 400
Checksum: 14f7f
Subsystem: 2
DLL characteristics: 8000
Size of stack reserve: 40000
Size of stack commit: 11000
Size of heap reserve: 100000
Size of heap commit: 1000
Loader flags: 0
Number of RVA and sizes: 10
Data directory size: 0
Data directory virtual size: 0
```

Slika 5.3 Ispis *PE* zaglavlja

Ispis *PE* zaglavlja organiziran je prema odgovarajućoj strukturi iz *winnt.h* datoteke. Polja u toj strukturi su nazvana dovoljno intuitivno pa su iskorištena pri ispisu.

Dodavanjem još i parametra „**–s**“ kod poziva programa dobit ćemo ispis o svim sekcijama *PE* datoteke.

```
SECTION HEADER [0]

Name: .text
Virtual address: 1000
Size of raw data: 7800
Pointer to raw data: 400
Pointer to relocations: 0
Pointer to line numbers: 0
Number of relocations: 0
Number of line numbers: 0
Characteristics: 60000020

SECTION HEADER [1]

Name: .data
Virtual address: 9000
Size of raw data: 800
Pointer to raw data: 7c00
Pointer to relocations: 0
Pointer to line numbers: 0
Number of relocations: 0
Number of line numbers: 0
Characteristics: c0000040

SECTION HEADER [2]

Name: .rsrc
Virtual address: b000
Size of raw data: 8a00
Pointer to raw data: 8400
Pointer to relocations: 0
Pointer to line numbers: 0
Number of relocations: 0
Number of line numbers: 0
Characteristics: 40000040
```

Slika 5.4 Ispis zaglavlja sekcija

U ovom slučaju (datoteka *notepad.exe*) postoje tri sekcije. Imena sekcija se ispisuju u tekstualnom obliku, iako u praktičnom smislu naziv nije bitan.

Ovi ispisi imaju svrhu proučavanja strukture izvršne datoteke. Neka polja ovih zaglavlja koriste se kao osnova za heurističko otkrivanje.

5.2.3 Pretraživanje datoteka

Za pretraživanje datoteka koriste se parametri „**–f**”, „**-d**” i „**–r**”. Parametrom „**–f**” i navođenjem putanje do datoteke, može se pretražiti bilo koji tip datoteke. Parametrom

„**–d**” i zadavanjem putanje do željenog direktorija pretražit će se samo datoteke za koje se pretpostavlja da imaju *PE* format. Filtriranje se vrši prema ekstenziji, i pretražuju se samo *.exe* i *.dll* datoteke. Dodavanjem opcionalnog parametra „**–r**”, jednako pretraživanje obavit će se i u svim poddirektorijima (rekurzivno).

Program koristi metode pretraživanja niske razine, i mogao bi se ubrojiti u pretraživače prve generacije. Za detekciju se koristi baza potpisa poznatog malware-a. Baza je pohranjena u tekstualnoj datoteci naziva *virdb.txt* koja se mora nalaziti u istom direktoriju kao i program.

Format zapisa potpisa *malware-a* je jednostavan. Svaki red datoteke sadrži podatke za jednu instancu *malware-a.*

Kako bi se baza potpisa mogla koristiti, svi zapisi moraju biti zapisani u formatu:

```
<naziv_malwarea>:<tip_pretrage>:<pocetak>:<kraj>:<scan_do_kraja_datoteke>:<potpis_malwarea>
```

Delimiteri polja zapisa su znakovi „**:**”. Naziv *malware-a* služi kao oznaka instance na koju se odnosi ovaj potpis. Tip pretrage označava radi li se o potpisu koji sadržava ili koji ne sadržava zamjenske znakove (*wildcards*). Oznaka „**1**” u tom polju označava potpis bez *wildcard* znakova, a oznaka „**2**” označava potpis sa *wildcard* znakovima. Umetanje oznake „**1**” u to polje za potpis koji sadržava *wildcard* znakove uzrokovat će pojavu da taj potpis neće biti pronađen u nijednoj datoteci.

Slijedeća tri polja označavaju koje dijelove datoteka treba pretražiti. Moguće je odrediti početak i kraj pretraživanja ukoliko je poznato koje adrese virus napada. To je prikladno za *prepending* viruse poznate veličine. Tada se može početi pretraživanje od početka (ili blizu početka) i prekinuti pretraživanje nakon neke adrese. Ukoliko je poznato da se radi o *appending* virusu, moguće je odrediti veličina od kraja datoteke koju treba pretražiti. Ta veličina se zadaje u `<scan_do_kraja_datoteke>`. Neka vrijednost (različita od „-1”) u tom polju poništava vrijednosti u prethodna dva polja. Negativna vrijednost u polju `<pocetak>` interpretira se kao pretraživanje od početka datoteke. Vrijednost „-1” u polju `<kraj>` označava pretraživanje do kraja datoteke (nije nužno poznavati veličinu datoteke). Navedena polja ne odnose se na pretraživanje memorije procesa. Kod pretraživanja memorije pretražuju se cijeli moduli. Moduli ne preslikavaju u memoriju sa jednakim razmacima između njihovih dijelova kao na disku stoga se vrijednosti udaljenosti iz zapisa potpisa ne mogu primjeniti kod pretraživanja memorije.

Potpis *malware-a* ima također specifičan format u kojem mora biti zapisan. Potpis u osnovi sadrži parove heksadecimalnih znamenki. Svaki par znamenki označava vrijednost u skupu [0, 255]. Ta vrijednost odgovara određenom konstantnom *byte-u* u binarnom kodu *malware-a*.

Osim toga potpisi mogu sadržavati jednostavne zamjenske znakove. Oni služe kao zamjena za dijelove *malware-a* koji mogu biti varijabilni. Zamjenski znakovi su slijedeći:

Tablica 5.2 Zamjenski znakovi za program *PEScanner*

?	Zamjenjuje samo jedan varijabilni *byte.*
*	Zamjenjuje bilo koliko varijabilnih *byte-ova.*
{n}	Zamjenjuje točno **n** varijabilnih *byte-ova.*
{n+}	Zamjenjuje **n** ili više varijabilnih byte-ova.
{n-}	Zamjenjuje najviše **n** varijabilnih *byte-ova.*

Ograničenje kod korištenja zamjenskih znakova je u tome da nije moguće koristiti više zamjenskih znakova zaredom. No u praktičnom smislu to nije ni potrebno jer se bilo koja kombinacija dva zamjenska znaka može predstaviti pomoću samo jednog zamjenskog znaka. Primjer je niz „**?***" koji znači preskakanje jednog znaka i preskakanja neodređenog broja znakova. Taj niz je istoznačan sa jednim znakom „***". Također, korištenje zamjenskih znakova na početku i na kraju potpisa nema praktičnu svrhu nije dozvoljeno.

Potrebno je obratiti pažnju na pravilan odabir tipa u zapisu *malware-a* u bazi potpisa. Obje vrste potpisa se čitaju istom funkcijom tako da neće doći do ozbiljne pogreške. No funkcionalnost programa će biti narušena. Korištenje oznake „**1**" kod potpisa koji sadržava zamjenske znakove učinit će taj potpis beskorisnim. To ne znači da program neće pokušati pronaći taj potpis unutar pretraživane datoteke. Pretraživanje bez pronalaska je vremenski gledano najlošiji slučaj i program uzalud troši vrijeme.

U drugom slučaju, moguće je navesti oznaku „**2**" u zapisu *malware-a* čiji potpis ne sadrži zamjenske znakove. U tom slučaju tokom pretraživanja potpis će bit uredno detektiran ukoliko postoji u datoteci. Nedostatak ovakvog zapisa nastaje zbog toga što pri pretraživanju pomoću potpisa koji ima zamjenske znakove program ne koristi brzi *Boyer-Moore* algoritam, i dolazi do bespotrebnog usporavanja pretraživanja.

Program ispisuje imena i putanje pretraživanih datoteka. Ukoliko nijedan potpis nije detektiran u datoteci, imena će jednostavno biti odvojena crtom.

```
...

Scanning File: s:\\PEscanner\apr1.exe


---------------------------------------------

Scanning File: s:\\PEscanner\PEscanner.exe

Error: Ne mogu otvoriti datoteku.
---------------------------------------------

Scanning File: s:\\PEscanner\test.exe


---------------------------------------------

Scanning File: s:\\PEscanner\test2.exe


---------------------------------------------

Scanning File: s:\ PEscanner\test3.exe


---------------------------------------------

...
```

Slika 5.5 Ispis pretraživanja

Ukoliko se određeni potpis pronađe unutar datoteke ispisat će slijedeća obavijest:

```
...
---------------------------------------------

Scanning File: Infected_file.exe

Found virus signature match:
      Signature:  Some.virus.signature
      Address:    38800
      FILE:       Infected_file.exe

...
```

Slika 5.6 Pronalazak inficirane datoteke

Obavijest ispisuje ime koje je zapisano u bazi potpisa. Drugi podatak je adresa unutar datoteke na kojoj se nalazi prvi *byte* prepoznatog virusa. Treći podatak je ispis imena inficirane datoteke.

5.2.4 Pretraživanje memorije

Program nudi funkciju pretraživanja memorije, koje se aktivira dodavanjem parametra „**-m**” kod pokretanja programa. Time se pokreće pretraživanje jednako onome nad datotekama, ali nad svim dijelovima memorije za koje trenutni korisnik dozvolu. Ukoliko ne postoji dozvola za dohvaćanje nekog procesa, ispisat će se poruka.

Ispis kod pretraživanja memorije izgleda ovako:

```
procesa ima: 45
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

PROCESS:<unknown> PID:0
FAIL: cannot open process handle
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

PROCESS:<unknown> PID:4
FAIL: cannot open process handle
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~


PROCESS: smss.exe (PID: 820)


[0] smss.exe (0x48580000)
      Base of Module: 0x48580000
      Entry point: 0x4858A4C8
      Size of image: 0x0000F000


[1] ntdll.dll (0x7C900000)
      Base of Module: 0x7C900000
      Entry point: 0x7C913156
      Size of image: 0x000B0000
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

...
```

Slika 5.7 Pretraživanje memorije

Prvi podatak koji se ispisuje je broj procesa koji se trenutno izvršavaju na *Windows* sustavu. Za svaki proces za koji je to moguće (zbog prava pristupa) ispisuje se ime procesa i njegov identifikacijski broj (*process identification – PID*). Svaki proces se sastoji od više modula. Za svaki od njih ispisuju se neki osnovni podaci. To su početna adresa modula, ulazna točka i veličina preslike (i*mage*) datoteke u memoriju. Struktura i organizacija memorijskog prostora biti će objašnjena kasnije. Pomoću ovog ispisa, moguće je vidjeti koje *.dll* datoteke koriste koji programi.

Svi moduli se pretražuju, i ukoliko se detektira neki od potpisa, ispisati će se poruka slična kao kod pretraživanja datoteka.

```
...

PROCESS: Infected_file.EXE (PID: 1684)


[0] Infected_file.EXE (0x01000000)
      Base of Module: 0x01000000
      Entry point: 0x0100739D
      Size of image: 0x00014000
Found virus signature match:
      Signature:  Some.virus.signature
      Address:    50064
      PID:  1684
      Name:       Infected_file.EXE
      Module:     Infected_file.EXE

...
```

Slika 5.8 Pronalazak inficiranog modula

5.2.5 Heurističko otkrivanje

Program sadržava funkcije koje provjeravaju pet pokazatelja koji mogu ukazivati na to da datoteka ima sumnjivu strukturu koja može biti posljedica infekcije virusom. Pokazatelji su opisani u poglavlju sa opisima heurističkih pokazateljima na *PE* datotekama. Heurističko otkrivanje može se izvršiti nad datotekama na tvrdom disku uz „obično" pretraživanje, ili nad modulima u memoriji. Postupak je jednak za oba slučaja, a aktivira se dodavanjem parametra „**-h**" kod pokretanja programa.

Provjeravaju se slijedeći pokazatelji:

- Je li ulazna točka programa u zadnjoj sekciji
- Koriste li se sumnjive kombinacije u karakteristikama sekcija
- Je li ulazna točka prije prve sekcije (infekcija zaglavlja)
- Odgovara li veličina izvršnog binarnog koda veličini u zaglavlju
- Odgovara li veličina datoteke (*image*) veličini u zaglavlju

```
...

Scanning File: some_file.exe

HEURISTICS:
--------------------------------
Entry point not in last section - OK!!!
Section characteristics OK!!!
Entry point not before first section - OK!!!
Size of code OK!!!
Size of image OK!!!
--------------------------------

...
```

Slika 5.9 Ispis kod heurističkog otkrivanja

5.3 Implementacija

5.3.1 Boyer-Moore algoritam

Antivirusni alati moraju raditi što brže kako ne bi smanjivali performanse sustava. Prvi korak prema ubrzanju je odabir odgovarajućeg algoritma koji se koristi za pretraživanje datoteka ili dijelova sustava. U praktičnom dijelu ovog rada koristi se algoritam kojeg su stvorili *R. Boyer* i *J.S. Moore* 1997. godine.

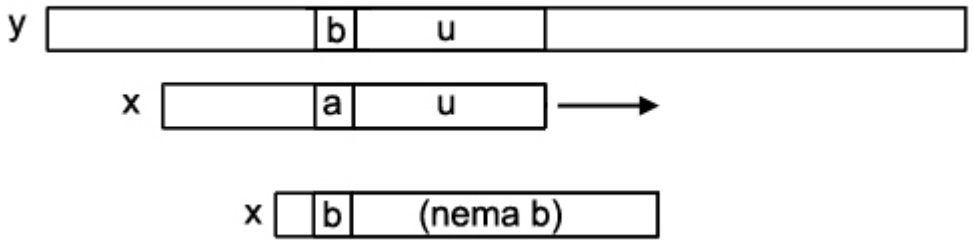
Posebnost algoritma je što pretprocesira traženi uzorak (u ovom slučaju potpis virusa), a ne niz unutar kojega se uzorak traži. Algoritam ima nisku složenost i učinkovitost mu se čak poboljšava sa većim duljinama uzoraka koji se traže.

Algoritam se temelji na dvije metode kojima se određuje pomak traženog uzorka u slučaju da se promatrani znak uzorka ne poklapa sa znakom pretraživanog niza. Usporedba se vrši sa desna na lijevo (od kraja uzorka prema početku). Te dvije metode se nazivaju heuristika dobrog sufiksa (*good sufix heuristics*) i heuristika lošeg znaka (*bad character heuristics*).

Neka se traženi uzorak označi kao niz **x[]** duljine **m**, a niz koji pretražujemo niz **y[]** duljine **n**. Znakovi se uspoređuju s desna na lijevo, a pozicija na nizu **y** se pomiče s lijeva na desno. Tako će prvi uspoređeni znakovi biti **x[m - 1]** i **y[m - 1]**(računato od nule). Kod pronalaska neodgovarajućih znakova, pozicija na nizu **y** se pomiče u desno za vrijednost **z** koja se određuje navedenim metodama. U drugoj iteraciji prvo će se uspoređivati **x[m - 1]** i **y[z + m - 1]**.

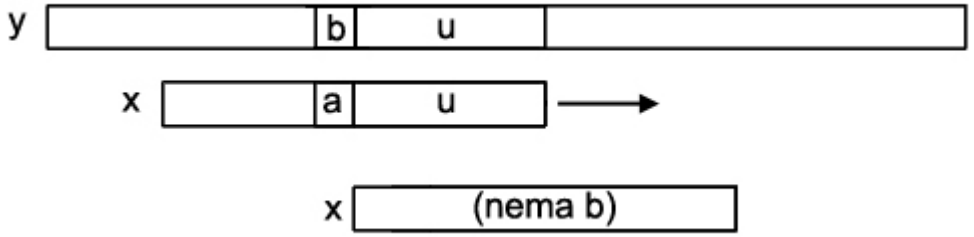
Heuristika dobrog sufiksa

Neka se nizovi **x** i **y** razlikuju na poziciji gdje vrijedi **x[i] = a** i **y[j + i] = b**. Podnizovi desno od tog mjesta se podudaraju, tj. vrijedi: ***x*[*i*+1 .. *m*-1] = *y*[*i*+*j*+1 .. *j*+*m*-1] = u**. Ova metoda pokušava pomaknuti **x** za toliko mjesta da se podudarajući podniz u **y** podudara sa najdesnijim podnizom iz **x** kojem ne prethodi **x[i] = a.**



Slika 5.10 Heuristika dobrog sufiksa (1)

Ukoliko takav podniz ne postoji na nekom mjestu osim na samom kraju niza **x** onda se niz **x** pomiče toliko udesno da se podudaraju najduži mogući prefiks niza **x** i sufiks niza ***y*[*i*+*j*+1 .. *j*+*m*-1]**.



Slika 5.11 Heuristika dobrog sufiksa (2)

Kako bi se mogao izvršiti takav pomak, potrebno je pretprocesirati niz **x**. Prvo se izvodi pretprocesiranje za slučaj da postoji podniz unutar niza **x** i na nekom mjestu osim na potpuno desnoj poziciji. Podudarajući sufiks se naziva granica. Izračunavaju se dodatni nizovi **f** i **s**. Niz **f[i]** sadrži pozicije najširih granica koji postoje u sufiksu niza **x** koji počinje na poziciji **i**. U **s[i]** se pohranjuje pomak koji se može izvršiti ukoliko dođe do nepodudaranja znakova na toj poziciji. Pseudokod ovog dijela pretprocesiranja izgleda ovako:

```
PretprocesiranjeGS1()
{
    int i=m, j=m+1;
    f[i]=j;
    while (i>0)
    {
        while (j<=m && x[i-1]!=x[j-1])
        {
            if (s[j]==0) s[j]=j-i;
            j=f[j];
        }
        i--; j--;
        f[i]=j;
    }
}
```

Slika 5.12 Pretprocesiranje kod metode dobrog sufiksa, prvi dio

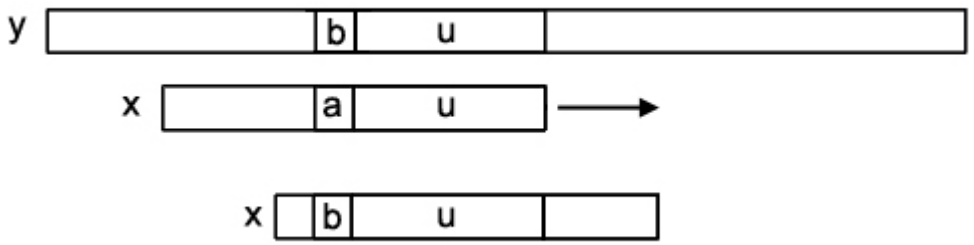
Nakon toga slijedi drugi dio pretprocesiranja za metodu dobrog sufiksa. Taj dio se odnosi na slučaj kada se podudaranje sa sufiksom niza **x** može pronaći jedino na početku niza **x**. Pseudokod tog dijela pretprocesiranja:

```
PretprocesiranjeGS2()
{
    int i, j;
    j=f[0];
    for (i=0; i<=m; i++)
    {
        if (s[i]==0) s[i]=j;
        if (i==j) j=f[j];
    }
}
```

Slika 5.13 Pretprocesiranje kod metode dobrog sufiksa, drugi dio

Heuristika lošeg znaka

U trenutku kada pri usporedbi znakova s desna na lijevo dođe do nepodudaranja, ovom metodom se također pokušava pomaknuti niz **x** što je više moguće u desno. Neka podudaranje nastane između znakova **x[i] = a** i **y[j + i] = b**. Ideja je pomaknuti niz **x** toliko udesno da se poklapaju znak **y[j + i]** i najdesnija pojava tog znaka u nizu **x**. Ukoliko ne postoji takva pojava, niz **x** će se pomaknuti toliko da stoji iza znaka **y[j + i]**.



Slika 5.14 Heuristika lošeg znaka

Pseudokod za ovu metodu:

```
PretprocesiranjeBC()
{
    char a;
    int j;

    for (a=0; a<velicina_abecede; a++)
        bc_shift[a]=-1;

    for (j=0; j<m; j++)
    {
        a=p[j];
        bc_shift [a]=j;
    }
}
```

Slika 5.15 Metoda lošeg znaka

Konačno računanje pomaka

Ukoliko dođe do nepodudaranja znakova pri usporedbi, koristi se povoljniji (veći) pomak, izabran između pomaka dobivenih pomoću dvije navedene metode. Pseudokod glavne funkcije algoritma izgleda ovako:

```
Boyer_Moore()
{
    int i=0, j;
    while (i<=n-m)
    {
        j=m-1;
        while (j>=0 && p[j]==t[i+j]) j--;
        if (j<0)
        {
            report(i);
            i+=s[0];
        }
        else
            i+=MAX(s[j+1], j-bc_shift[t[i+j]]);
    }
}
```

Slika 5.16 Glavna funkcija Boyer-Moore algoritma

5.3.2 Pretraživanje sa zamjenskim (*wildcard*) znakovima

Kod pretraživanja sa zamjenskim znakovima u *PEscanner-u* se ne koristi *Boyer-Moore* algoritam. Algoritam koji implementira pretraživanje sa zamjenskim znakovima zahtijeva u nekim slučajevima velik broj usporedbi (neuspješnih) što ga čini puno sporijim od *Boyer-Moore* algoritma.

PEscanner uzima potpis virusa iz datoteke i parsira ga. Parove heksadecimalnih znakova (jedan *byte*) pohranjuje se u polje *integera*. Time dio svakog elementa polja zapravo ostaje neiskorišten. No, kada program parsirajući potpis dođe do zamjenskog znaka, tada se u polje pohranjuje njegov „kod”. Kodovi zamjenskih znakova u memoriji su brojevi veći od maksimalne vrijednosti koja se može zapisati u jedan *byte*. Ako zamjenski znak ima brojčani argument, taj argument se u memoriji postavlja na mjesto iza „koda” zamjenskog znaka.

Kod pretraživanja izvršavaju se dvije ugnježdene petlje. Vanjska petlja pomiče početnu adresu pretraživanja prema zadanim veličinama iz datoteke s potpisima virusa. Unutarnja petlja usporedno pomiče pokazivače na uspoređivane znakove promatranog niza i potpisa. Ukoliko su znakovi jednaki pomiču se oba pokazivača unutarnje petlje. Ukoliko su različiti, a znak potpisa nije zamjenski znak (veći od 255 – jedan *byte*), pokazivači unutarnje petlje se postavljaju na nulu, a vanjska petlja se pomiče za jedan.

Implementacije zamjenskog znaka koji označava jedan varijabilan znak, i onaj koji označava **n** varijabilnih znakova su jednostavne, pokazivač na promatrani niz se povećava za jedan ili za **n**. Zamjenski znakovi kod kojih se nezna koliko točno znakova zamjenjuju moraju se implementirati malo drugačije. Znak „*****” potpis dijeli na dva dijela koji se oba moraju pronaći unutar promatranog niza, ali mogu biti odvojeni i proizvoljno

udaljeni. Znakovi „**{n-}**" i „**{n+}**" su vrlo slični tome, no sadržavaju maksimalnu odnosno minimalnu udaljenost dva navedena podniza. Kada algoritam dođe do jednog od ovih znakova, rekurzivno se poziva funkcija za usporedbu sa modificiranim parametrima za početnu i završnu adresu u nizu koji se provjerava. Znakovi „*****"i „**{n+}**"mijenjaju adresu na kojoj počinje pretraživanje u idućoj razini rekurzije, dok znak „**{n-}**" mijenja adresu početka i kraja. U svim slučajevima obavljaju se provjere da se ne prekorače granice promatranog niza.

```
WildCard(p_niz,k_niz,p_potpis,k_potpis){
      for(i=p_niz;i<k_niz;i++){
            for(j=p_potpis,m=0;j<k_potpis;j++,m++){
            ako niz[i+m]==potpis[j] nastavi;
            ako potpis[j]==“?“ nastavi;
            ako potpis[j]==“{n}“ {m=m+n; nastavi;}
            ako potpis[j]==“*“ {
                  retval=WildCard(i+m+1,k_niz,j+1,k_potpis);
                  vrati retval;
                  }
            ako potpis[j]==“{n+}“ {
                  m=m+n;
                  retval=WildCard(i+m+1,k_niz,j+1,k_potpis);
                  vrati retval;
                  }
            ako potpis[j]==“{n-}“ {
                  retval=WildCard(i+m+1,i+n,j+1,k_potpis);
                  ako(retval!=-1) vrati retval
                  inace break;
                  }
            }
      }
}
```

Slika 5.17 Pseudokod pretraživanja sa zamjenskim znakovima

5.3.3 Pretraživanje memorije u korisničkom načinu rada

Sustav virtualne memorije na *Windows NT*

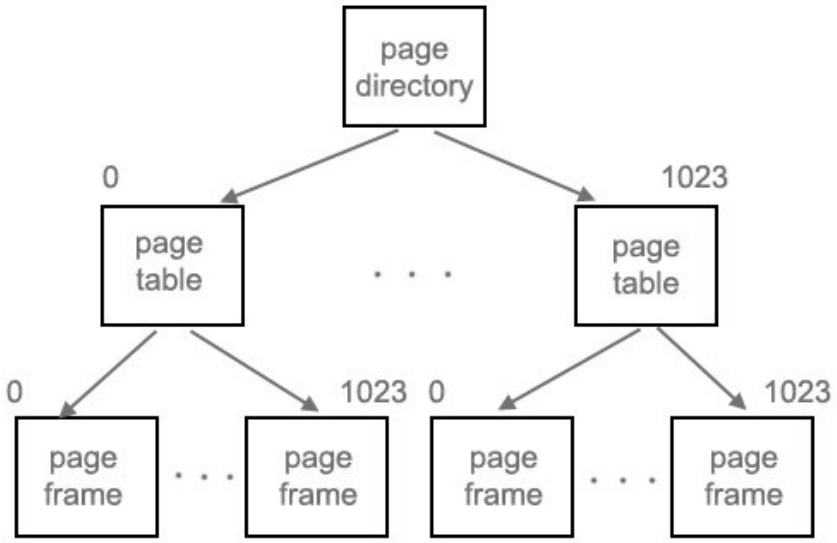
Windows NT (i operacijski sustavi razvijeni na toj osnovi) sadržava sustav virtualne memorije, kakav nije bio dostupan na starijim *Windows* operacijskim sustavima. Svaki proces može dohvatiti četri GB adresnog prostora pomoću 32-bitnih adresa. Nižih dva GB je dostupno za korištenje procesu, dok je viših dva GB rezervirano za operacijski sustav.

Virtualnim memorijskim prostorom se bavi *Virtual Memory Manager (VMM).* Taj dio operacijskog sustava omogućava da se koristi prividno linearan memorijski prostor koji može biti i veći od stvarne količine fizičke radne memorije. Rad *VMM-a* se temelji na korištenju konstantno velikih dijelova memorije zvanih stranice (*pages*). Svaka stranica

je velika 4096 *byte-ova* bez obzira na način upotrebe te stranice. Stvarne radne memorije može biti nedovoljno za izvršavanje svih procesa, pri čemu se *VMM* brine za zamjenu stranica iz memorije sa stranicama pohranjenim na tvrdom disku u tzv. *pagefile-ovima.*

Za dohvat memorijskih lokacija koriste se 32-bitne virtualne adrese. Dijelovi te adrese koriste se za određivanje stvarne adrese koju treba dohvatiti.

Memorijska struktura može se prikazati strukturom stabla. Svaki proces ima svoj direktorij stranica (*page directory*) u kojem se nalazi 1024 zapisa (*page directory entry – PDE*) Prvih deset bitova virutalne adrese se odnosi na jedan od tih zapisa. *PDE* pokazuje na tablicu stranica (*page table*) koja također sadrži 1024 zapisa (*page table entry – PTE*) koji pokazuju na neku stranicu (*page frame*). Drugih deset bitova se odnosi na jedan od zapisa iz *PTE*. U svakom *page frame-u* postoji 4096 *byte-ova* od kojih je svaki po jedna memorijska lokacija. Dohvaćanje jednog od njih se vrši pomoću zadnjih 12 bitova virtualne adrese [11].



Slika 5.18 Organizacija memorije

Ovakav sustav omogućava da svaki proces ima vlastiti memorijski prostor, što onemogućava slučajno ili namjerno upadanje u tuđi memorijski prostor. Takav pristup otežava korištenje zajedničke memorije u odnosu na starije operacijske sustave. Gornjih dva GB virtualnog memorijskog prostora rezervirani su za operacijski sustav. U tom prostoru se nalaze *driveri* i ostale komponente operacijskog sustava koji su dostupni svakom procesu.

Korištenje *Win32 API* funkcija za pretraživanje procesa

U programu se koristi eksplicitno dohvaćanje *.dll* datoteke koja sadrži *API* funkcije potrebne za pretraživanje procesa. Datoteka koja sadrži te funkcije je *psapi.dll*, i standardni je dio *Windows NT* (i srodnih) sustava. Potrebna biblioteka dohvaća se na slijedeći način:

```
...
HINSTANCE      hInstLib  = NULL;
hInstLib = LoadLibraryA("PSAPI.DLL");
...
```

Slika 5.19 Eksplicitno dohvaćanje .dll datoteke

Sada je moguće dohvaćati funkcije koje su sadržane u *psapi.dll.* Kako bi bilo moguće pretražiti memorijske prostore ostalih procesa na sustavu, potrebno ih je na neki način enumerirati. Svaki proces ima svoj identifikacijski broj (*process ID – PID*). Slijedeći primjer pokazuje eksplicitno dohvaćanje funkcije iz datoteke *psapi-dll*. Funkcija koju dohvaćamo koristi se za dohvaćanje *PID-ova* svih procesa na sustavu.

```
...
BOOL (WINAPI *lpfEnumProcesses)(DWORD *, DWORD, DWORD *);
lpfEnumProcesses = (BOOL (WINAPI *)(DWORD *, DWORD, DWORD*))
GetProcAddress(hInstLib, "EnumProcesses");
...
if ( !lpfEnumProcesses( aProcesses, sizeof(aProcesses), &cbNeeded ) )
        return 0;
...
```

Slika 5.20 Funkcija EnumProcesses() za enumeraciju procesa

Pozivanjem te funkcije u polju *aProcesses[]* će se nalaziti svi *PID-ovi* procesa koji se trenutno izvršavaju na sustavu. Broj procesa izračuna se kao **cbNeeded** / **sizeof(DWORD)**.

Identifikacijski broj procesa potreban je kako bi se pozivanjem određenih *API* funkcija dobio *handle* na svaki proces. To se izvršava pomoću funkcije koja je dio *Kernel32.dll* biblioteke:

```
...
HANDLE hProcess = OpenProcess( PROCESS_QUERY_INFORMATION |
       PROCESS_VM_READ,
       FALSE, processID );
...
```

Slika 5.21 Otvaranje *handle-a* procesa pomoći *PID-a*

Slijedeći korak je enumeracija svih modula unutar procesa. Moduli predstavljaju sve izvršne datoteke koje proces koristi. To uključuje izvršnu *.exe* datoteku koju npr. pokrećemo kao aplikaciju i sve .dll datoteke koje ta aplikacija koristi. Moduli unutar procesa mogu se vidjeti kroz ispis programa. Kada budu dostupni *handle-ovi* na sve module unutar procesa, može se pomoću određenih *API* funkcija pristupiti njihovoj memoriji.

```
...
BOOL (WINAPI *lpfEnumProcessModules)(HANDLE, HMODULE *, DWORD, LPDWORD);
DWORD (WINAPI *lpfGetModuleBaseName)(HANDLE, HMODULE, LPTSTR, DWORD);
DWORD (WINAPI *lpfGetModuleFileNameEx)(HANDLE, HMODULE, LPTSTR, DWORD);
BOOL (WINAPI * lpfGetModuleInformation)(HANDLE, HMODULE,
LPMODULEINFO,DWORD);

lpfEnumProcessModules = (BOOL (WINAPI *)(HANDLE, HMODULE *,DWORD,
LPDWORD))GetProcAddress(hInstLib, "EnumProcessModules");

lpfGetModuleFileNameEx = (DWORD (WINAPI *)(HANDLE, HMODULE, LPTSTR,
DWORD))GetProcAddress(hInstLib, "GetModuleFileNameExA");

lpfGetModuleBaseName = (DWORD (WINAPI *)(HANDLE, HMODULE,
 LPTSTR, DWORD)) GetProcAddress(hInstLib, "GetModuleBaseNameA");

lpfGetModuleInformation = (BOOL (WINAPI *)(HANDLE, HMODULE,LPMODULEINFO,
DWORD)) GetProcAddress(hInstLib, "GetModuleInformation");
...
```

Slika 5.22 Funkcije za dohvaćanje informacija o modulima

Ove funkcije su bitne radi općenitih informacija o modulima poput imena modula, ali za dohvaćanje početne adrese modula. Pomoću početne adrese i veličine *image-a* moguće je zaključiti koji dio memorije se pretražuje. Memorija se čita na slijedeći način:

```
...
MODULEINFO mdInfo;
...
lpfGetModuleInformation(hProcess,hMods[j],&mdInfo,sizeof(MODULEINFO));
...
lpMem=(int)mdInfo.lpBaseOfDll;
...
buf=(char *)malloc( mdInfo.SizeOfImage * sizeof(char));
...
ReadProcessMemory(hProcess,(void*)lpMem,buf,mdInfo.SizeOfImage,&buf_read
);
...
```

Slika 5.23 Čitanje memorijskog prostora drugog procesa

Nakon ovog dijela koda sadržaj ciljanog modula nalazi se u memorijskom prostoru programa za pretraživanje u obliku običnog polja. Korištenje funkcija koje pristupaju drugim procesima i njihovim modulima zahtijevaju određene razine pristupa na *Windows* sustavu.

Kada je pročitan modul iz drugog procesa, moguće ga je tretirati kao izvršnu datoteku čiji je sadržaj pročitan u memoriju. Na taj način može se primjeniti i pretraživanje usporedbom (*Boyer-Moore,* zamjenski znakovi) i koristeći heurističke pokazatelje u zaglavlju izvršne datoteke, koje ostaje sačuvano kod mapiranja datoteke u memoriju.

Kod pretraživanja svih procesa, u programu se enumeriraju svi moduli koje promatrani proces koristi. Pri tome se pretražuju i viših dva GB virtualnog adresnog prostora tog procesa, koji sadrže sistemske module. Operacijski sustav nudi većem broju procesa da koristi iste module (drivere...) stoga dolazi do toga da neki sistemski moduli budu pretraživani više puta. Za izbjegavanje tog problema bilo bi potrebno izgraditi određeni sustav prepoznavanja već pregledanih modula.

Sve *API* funkcije koje su korištene su opisane u službenoj dokumentaciji [12].

5.3.4 Heuristički pokazatelji

Kroz primjere funkcija koje provjeravaju heurističke pokazatelje jednostavno je pokazati kako, i pomoću kojih polja se vrše navedene provjere.

```
...
int first_section_start=this->scan_file->image_section_header[0]-
>PointerToRawData;

int entry_point=this->scan_file->image_nt_headers-
>OptionalHeader.AddressOfEntryPoint;

if(entry_point<first_section_start)
{
   return true;
}
return false;
...
```

Slika 5.24 Provjera infekcije zaglavlja

Iz zaglavlja prve(nulte) sekcije može se vidjeti njena početna adresa iz polja *PointerToRawData.* Ukoliko je adresa *AddressOfEntryPoint* manja od navedene adrese, postoji mogućnost infekcije *PE* zaglavlja.

```
...
int entry_point=this->scan_file->image_nt_headers-
>OptionalHeader.AddressOfEntryPoint;
int br_sect=this->scan_file->image_nt_headers-
>FileHeader.NumberOfSections;
int last_section_start=this->scan_file->image_section_header[br_sect-1]-
>PointerToRawData;
if(entry_point>last_section_start)
{
   return true;
}
return false;
...
```

Slika 5.25 Provjera pokazuje li ulazna točka u zadnju sekciju

Slično kao i u prethodnom primjeru, provjerava se je li adresa u *AddressOfEntryPoint* veća od adrese početka zadnje sekcije (*PointerToRawData*).

Ukupna količina koda zapisana je u opcionalnom zaglavlju *PE* datoteke, u polju *SizeOfCode*. Stvarna količina koda se računa iteracijom kroz zaglavlja sekcija. Veličina svake sekcije (*SizeOfRawData*) se dodaje zbroju ukoliko karakteristika sekcije pokazuje da se ta sekcija može izvršavati. Veličina *IMAGE_SCN_MEM_EXECUTE* zapisana je unutar *WINNT.h* datoteke.

```
...
DWORD chr;
int CodeSize=0;
for(int i=0;i<this->scan_file->image_nt_headers-
>FileHeader.NumberOfSections;i++)
{
      chr=this->scan_file->image_section_header[i]->Characteristics;
      if(chr & IMAGE_SCN_MEM_EXECUTE)
      {
           CodeSize=this->scan_file->image_section_header[i]-
      >SizeOfRawData;
      }
}

if(CodeSize!=this->scan_file->image_nt_headers-
>OptionalHeader.SizeOfCode)
{
   return true;
}
return false;
...
```

Slika 5.26 Provjera veličine koda

```
...
int ImageSize=0;
int left=0;
int alignment=this->scan_file->image_nt_headers-
>OptionalHeader.SectionAlignment;
int br_sect=this->scan_file->image_nt_headers-
>FileHeader.NumberOfSections;

ImageSize+=this->scan_file->image_section_header[br_sect-1]-
>VirtualAddress;

ImageSize+=this->scan_file->image_section_header[br_sect-1]-
>SizeOfRawData;

left=alignment - (ImageSize % alignment);
ImageSize+=left;

if(ImageSize!=this->scan_file->image_nt_headers-
>OptionalHeader.SizeOfImage)
{
   return true;
}
return false;
...
```

Slika 5.27 Provjera veličine datoteke

Veličina izvršne datoteke zapisana je u *PE* zaglavlju u polju *SizeOfImage*. Ta veličina se uspoređuje sa veličinom izračunatom tako da se adresi zadnje sekcije doda veličina zadnje sekcije, i dobiveni rezultat se zaokruži prema gore na višekratnik broja zapisan u *OptionalHeader.SectionAlignment.*

```
...
DWORD chr;
for(int i=0;i<this->scan_file->image_nt_headers-
>FileHeader.NumberOfSections;i++)
{
   chr=this->scan_file->image_section_header[i]->Characteristics;
   if(chr & IMAGE_SCN_MEM_WRITE) //cout<<"EXECUTABLE\n";
   {
      if(!(chr & IMAGE_SCN_MEM_READ))
      {
         return true;
      }
      if(chr & IMAGE_SCN_MEM_EXECUTE)
      {
         return true;
      }
   }
}
return false;
...
```

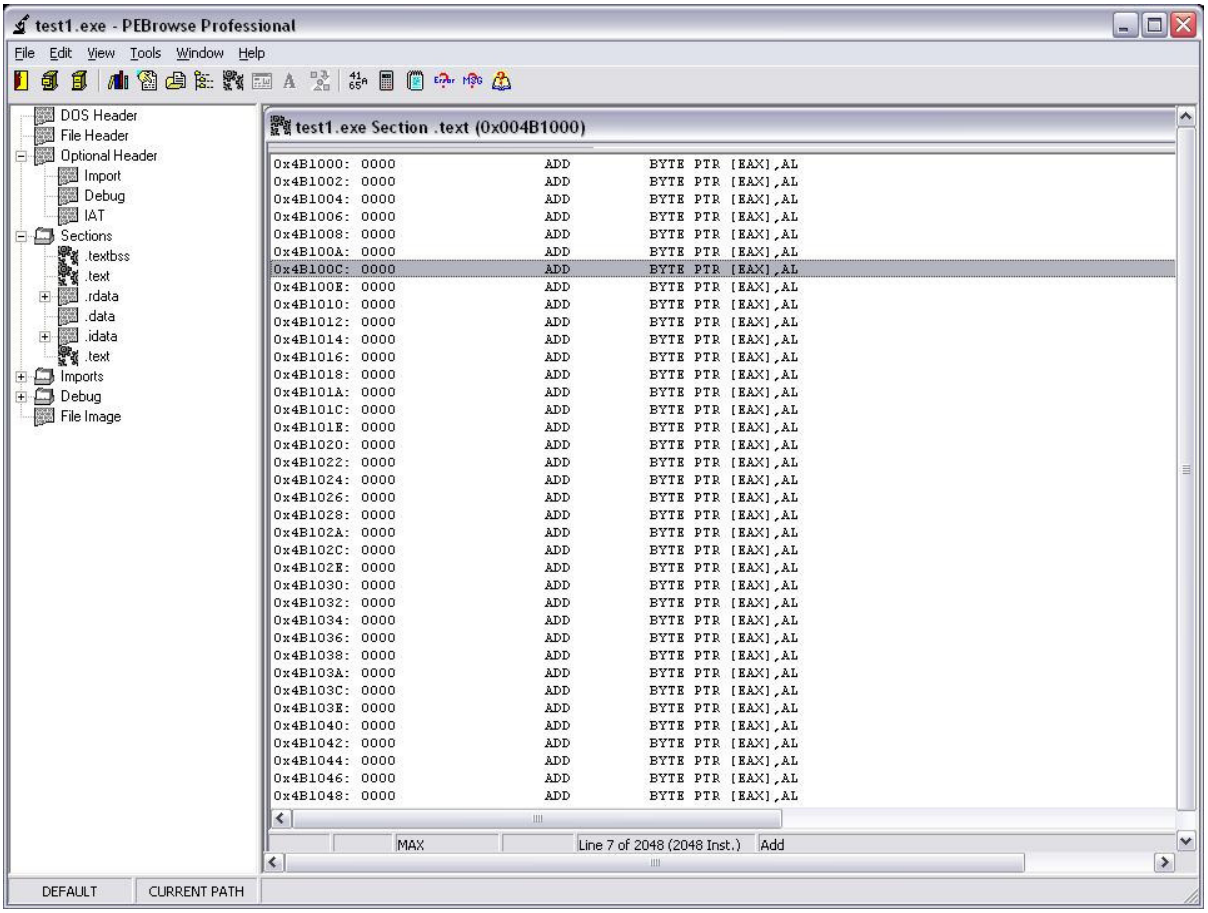
Slika 5.28 Provjera karakteristika sekcija

Provjeravaju se dvije kombinacije sumnjivih karakteristika sekcija. U oba slučaja je sekcija označena za pisanje. Ukoliko se iz iste sekcije ne može čitati, to se može smatratu sumnjivim pokazateljom. Također, ako je sekcija po kojoj se može pisati ujedno označena za izvršavanje, i to može biti pokazatelj za moguću prisutnost virusa.

5.4 Testiranje programa

5.4.1 Testiranje pretraživanja pomoću potpisa

Pretraživanje pomoću potpisa ne mora se odnositi na stvarni *malware*. Moguće je pretraživati datoteke pomoću potpisa neovisno što taj potpis logički sadržava. Testiranje pretraživača izvršeno je uzimanjem dijela koda iz običnog (bezopasnog) programa. Taj dio koda se koristi kao potpis. Testni potpis je izvađen iz programa *Notepad.exe*. Značenje dijela koda nije proučavano, no uzet je uzorak od 400-njak *byte-ova* kako bi se izbjeglo da takav uzorak bude nađen u nekom drugom programu. Za ekstrakciju potpisa iz testne datoteke korištena je besplatna inačica *PEBrowse Professional* programa.



Slika 5.29 Program *PEBrowse Professional*

PEBrowse program omogućava pregled *PE* datoteka. Pregled obuhvaća ispis vrijednosti polja *PE* zaglavlja, pregled sekcija u heksadecimalnom i disassembliranom obliku. Na taj način je lako iz datoteke kopirati niz heksadecimalnih vrijednosti potrebnih za potpis, iz sekcije koja sadržava binarni kod, a u ovom slučaju ima naziv *.text*.

```
Notepad.Signature:1:d5955900d8975b00db9e6500d79a650090776000e68d3300e695
4500db9c5e00e9ac7100dfa66e00e4b27f005f5c5900dd9b5400e09f5800e3a35b00e7a9
6500e1a46500f1b36d00e5a86900a68a6c00ffebd500837b7200e8a65400e3a35400e7a8
5e00ecaf6600ffc07500ffc17800ffc27800f6ba7400d0a47100c29c6e00ffdaac00ffe3
c100eaa95600ecac5a00c9924d00f0b05e00d49d5300ffbd6500f3b46000febe6a00f0b3
6500ffc07100ffc67b00ffc982009a876f00ffe8cb00fff6ea007f7b7600f0b05300f4b3
56
```

Slika 5.30 Testni potpis bez zamjenskih znakova

Za testiranje pretraživanja metodom sa zamjenskim znakovima, korišten je isti potpis kojem su određeni dijelovi zamijenjeni zamjenskim znakovima.

```
Notepad.WildSig:2:d5955900d8975b00db9e6500?9a650090*e68d3300e6954500db9c5e0
0{3}00dfa66e00e4b2{3-
}005f5c5900dd9b{3+}5800e3a35b00e7a96500e1a46500f1b36d00e5a86900a68a6c00ffeb
d500837b7200e8a65400e3a35400e7a85e00ecaf6600ffc07500ffc17800ffc27800f6ba740
0d0a47100c29c6e00ffdaac00ffe3c100eaa95600ecac5a00c9924d00f0b05e00d49d5300ff
bd6500f3b46000febe6a00f0b36500ffc07100ffc67b00ffc982009a876f00ffe8cb00fff6e
a007f7b7600f0b05300f4b356
```

Slika 5.31 Testni potpis sa zamjenskim znakovima

Pokretanjem pretraživanja diska i memorije rezultira ispisom obavjesti da su u datoteci (ili procesu) *Notepad.exe* pronađena oba potpisa.

```
PROCESS: NOTEPAD.EXE (PID: 356)


[0] NOTEPAD.EXE (0x01000000)
      Base of Module: 0x01000000
      Entry point: 0x0100739D
      Size of image: 0x00014000
Found virus signature match:
      Signature:  Notepad.Signature
      Address:    50064
      PID:  356
      Name:       NOTEPAD.EXE
      Module:     NOTEPAD.EXE
Found virus signature match:
      Signature:  Notepad.WildSig
      Address:    50064
      PID:  356
      Name:       NOTEPAD.EXE
      Module:     NOTEPAD.EXE
```

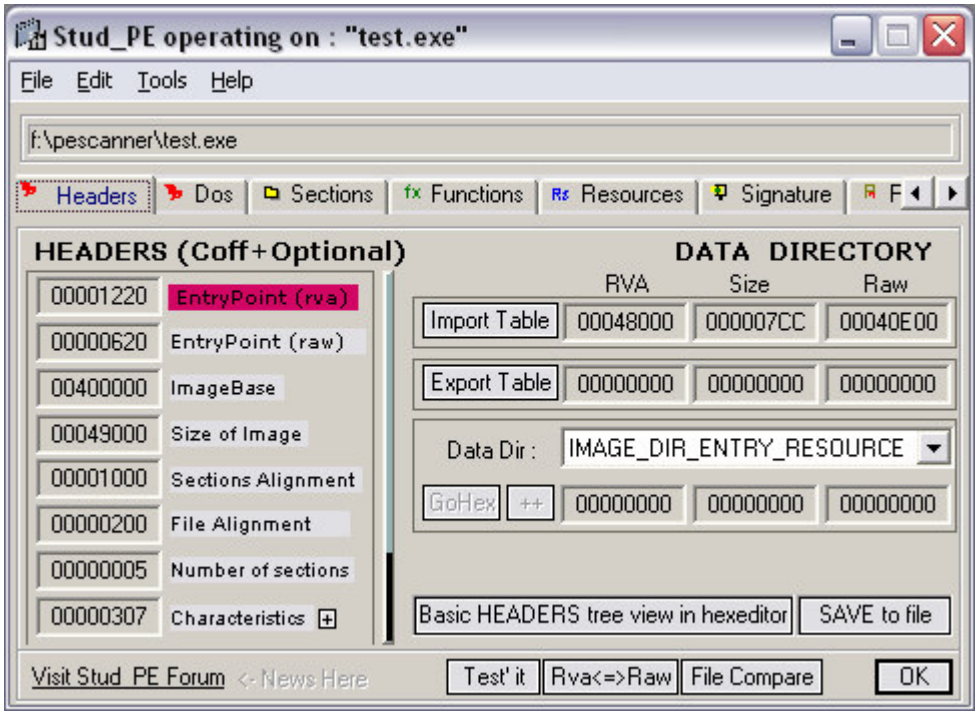
Slika 5.32 Uspješan pronalazak potpisa u procesu

5.4.2 Testiranje heurističkih pokazatelja

Za testiranje heurističkih pokazatelja potrebno je preurediti *PE* datoteku tako da ima svojstva koja mogu biti posljedica infekcije virusom. Testiranje pomoću stvarno inficiranih datoteka zahtjevalo bi korištenje postojećih virusa koji stvaraju opisane promjene u *PE* strukturi, što je teško izvedivo bez detaljnog poznavanja ponašanja stvarnih virusa. Stvaranje vlastitih virusa koji bi imali takva svojstva ne smatra se etički prihvatljivim.

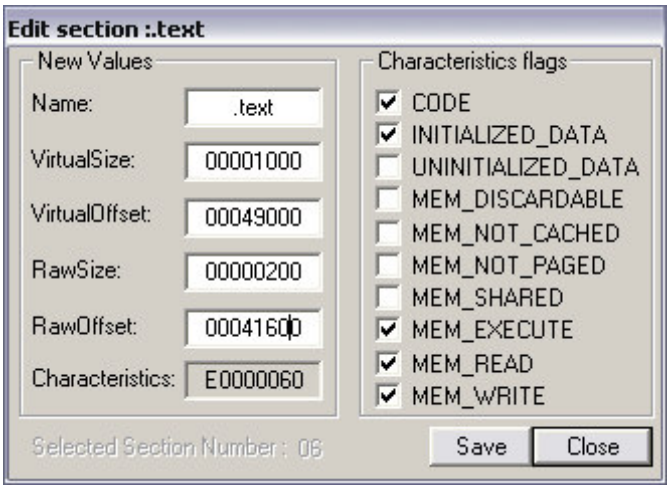
Program *Stud_PE* omogućava mijenjanje i pohranjivanje novih vrijednosti u zaglavlju *PE* datoteke. Osim toga, postoji mogućnost dodavanja novih sekcija.

Promjenom vrijednosti u nekim poljima *PE* zaglavlja simulira se inficirana datoteka. Tako modificirani program neće raditi, ali za testiranje to nije niti bitno.



Slika 5.33 Program *Stud_PE*

PE datoteci dodana je još jedna sekcija. Veličina dodane sekcije je odabrana 1000 *byte-ova.* Sekcija je dodana na kraj datoteke, poput *appending* virusa. Pri tome polja koja opisuju količinu koda i veličinu datoteke nisu promijenjena na nove stvarne vrijednosti. Karakteristika sekcije sadržava vrijednost koja opisuje da je sekcija dostupna za čitanje, pisanje i da sadrži kod. Ulazna točka programa se postavlja na početak dodane sekcije. Time su postavljeni četiri od pet heurističkih pokazatelja koji se provjeravaju praktičnim dijelom ovog rada (nemoguće je ujedno postaviti ulaznu točku ispred prve sekcije). Tako izmijenjena datoteka je pohranjena na disk.



Slika 5.34 Postavljanje karakteristike sekcije

Pokretanje programa nad testnom datotekom dobiveni su predviđeni rezultati. Rad programa testiran je jednako uspješno i sa ulaznom točkom postavljenom prije prve sekcije.

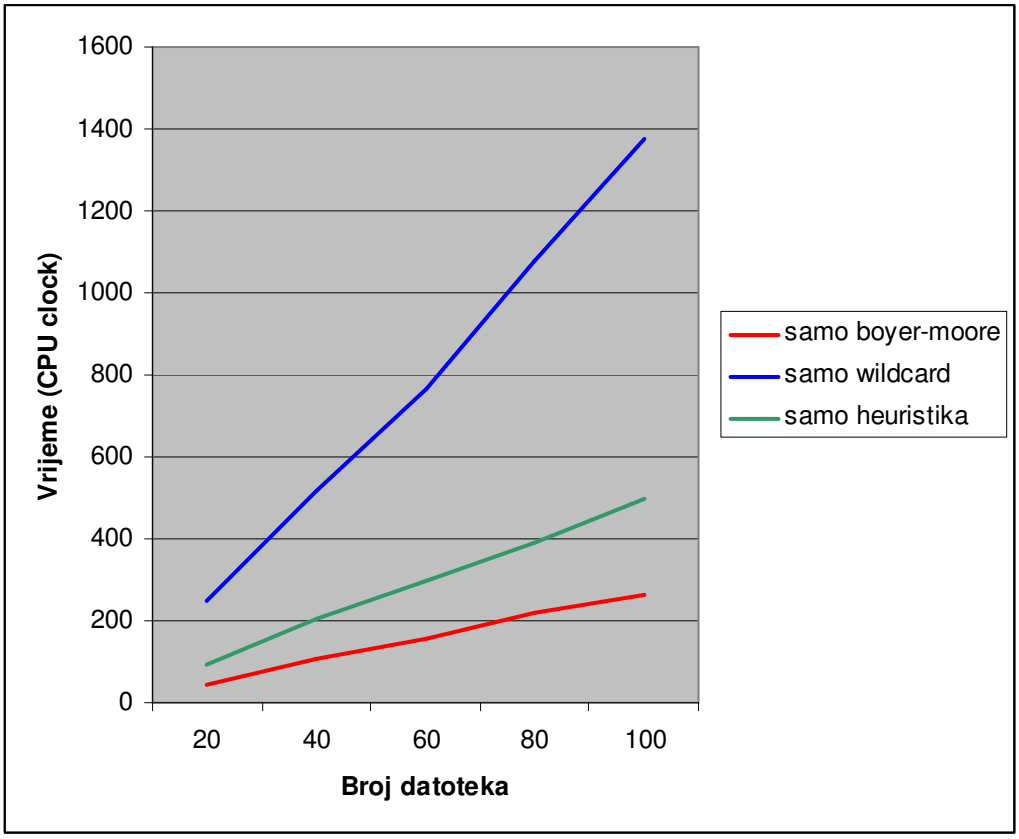
```
Scanning File: test1.exe

HEURISTICS:
--------------------------------
Entry point in last section!!!
Suspicious section characteristics!!!
Entry point not before first section - OK!!!
Incorrect size of code!!!
Incorrect image size!!!
--------------------------------
```

Slika 5.35 Testiranje provjere heurističkih pokazatelja

5.4.3 Brzina izvođenja

Testirana je brzina pretraživanja datoteka. *PEscanner* pretražuje pomoću tri glavne komponente: *Boyer-Moore* algoritam, pretraživanje sa zamjenskim znakovima i otkrivanje heurističkim pokazateljima. Brzina naravno ovisi o računalu na kojem se izvodi, ali promatranjem međusobnih odnosa može se vidjeti kako pojedina komponenta utječe na vrijeme izvođenja. Izvršeno je više mjerenja u ovisnosti o broju pretraživanih datoteka. Pretraživana datoteka je veličine 504 KB. Kao potpisi virusa se koriste već opisani potpisi izvađeni iz datoteke *notepad.exe.* U promatranoj datoteci ne pronalazi se taj potpis, i uzima se najgori slučaj, kada se pretražuje cijela datoteka. Potpis sa

zamjenskim znakovima sadrži znakove „*” i „{n+}” koji oduzimaju najviše vremena, no potpis je prikladan za usporedbu.



Slika 5.36 Trajanje pretraživanja

5.5 Razmatranje o nadogradnji

5.5.1 Općenita razmatranja

Osim tehnika koje su implementirane u praktičnom dijelu ovog rada postoji cijeli niz pristupa koji se smatraju uobičajenim u modernim antivirusnim programima. No pretraživanje pomoću potpisa je i dalje jedna od najčešće korištenih metoda u antivirusnim programima. Veliku ulogu u tom pristupu ima brzina izvođenja. *PEscanner* pretražuje datoteke i procese na vrlo niskom nivou, sa jedinim ubrzanjem u vidu određivanja područja datoteka koja se pretražuju. *PE* struktura se ne upotrebljava za određivanje tih adresa. Kod pretraživanja memorije neki moduli koji su dijelovi operacijskog sustava, pretražuju se više puta, što dodatno usporava program. Prvi cilj nadogradnje ovog programa je detaljnije specificiranje područja pretraživanja (npr. dodavanje pretraživanja od ulazne točke). Isto ubrzanje nužno je za pretraživanje memorije, gdje se *PE* struktura mora iskoristiti za odabir dijelova memorije koji će biti pretraživani, a koji izostavljeni. Korištenje na taj način poboljšanih performansi programa

zahtijeva poznavanje specifičnosti svakog pojedinog virusa (kako bi se napravio odgovarajući potpis)

Osim toga, iako se koristi relativno brzi, *Boyer-Moore* algoritam uspoređivanja, ako se uzme u obzir da baze potpisa virusa sadržavaju više desetaka tisuća potpisa, to neće biti dovoljno brz postupak da ne utječe na performanse računala. Potrebno je dodatno ubrzati proces pretraživanja. Jedna od predloženih metoda je korištenje funkcija za izračunavanje sažetka (*hash* funkcije).

5.5.2 Ubrzavanje pretraživanja pomoću *hash* funkcija

U nekim radovima [13] se razmatra korištenje *hash* funkcija i *bloom filtera* za ubrzavanje procesa pretraživanja pomoću potpisa. *Bloom filter* je niz bitova koji sadržava rezultate nekoliko različitih *hash* funkcija nad skupom uzoraka. *Bloom filter* ima onoliko bitova koliko ima mogućih rezultata korištenih hash funkcija. Ukoliko se hash funkcijom nad nekim uzorkom (u ovom slučaju potpisu ili dijelu potpisa virusa) dobije **n**, tada se **n-ti** bit *bloom filtera* postavlja na „1" (inicijalno je vrijednost „0"). Takav filter može biti dovoljno malen čak da stane u priručnu (*cache*) memoriju. Time se gubi puno manje ciklusa centralne procesne jedinice koji se inače troše na dohvat iz *RAM* memorije.

Ovaj pristup se temelji na korištenju „loših" i brzih funkcija za računanje sažetaka. Loša *hash* funkcija ima veću vjerojatnost davanja istog rezultata za različite ulazne uzorke, ali su brzine takvih funkcija redovite veće od „dobrih" *hash* funkcija.

Mehanizam ove metode je sljedeći: *Bloom filter* ima duljinu **N** bitova. Nad prvih **β** *byte-ova* svakog pojedinog potpisa virusa (nazvanih **a**) izvršava se **k** različitih *hash* funkcija $\mathbf{h_1(a), h_2(a), h_3(a)... h_k(a)}$. Vrijednosti rezultata se sve nalaze u intervalu **[1,N]**. Bitovi *bloom filtera* na pozicijama $\mathbf{h_1(a), h_2(a), h_3(a)... h_k(a)}$ se postavljaju na 1.

U procesu pretraživanja, nad svakim dijelom promatranog objekta (niz *byte-ova*, datoteka, dio sustava...) duljine **β**, (nazvanom **b)** izvršavaju se redom funkcije $\mathbf{h_1(b), h_2(b), h_3(b)... h_k(b)}$. Ukoliko funkcija $\mathbf{h_i(b)}$ da rezultat **x**, i u *bloom filteru* se pod rednim brojem **x** nalazi „1", izvršava se funkcija $\mathbf{h_{i+1}(b)}$. Ukoliko se pod tim rednim brojem nalazi „0", nije potrebno nastavljati izvršavanje $\mathbf{h_{i+1}(b)... h_k(b)}$, već se prelazi na slijedeći *byte* promatranog niza.

Izvršavanje svih **k** funkcija, i pozitivna usporedba sa *bloom filterom* znak je da je potrebno izvršiti točno prepoznavanje potpisa. Korištenje *hash* funkcija kako je opisano nije dovoljno za davanje sigurnog rezultata. Jedinica u *bloom filteru* može biti produkt krive *hash* funkcije (ne one koju provjeravamo) ili hash funkcija može dati isti rezultat za različite ulazne nizove. To će prouzročiti lažno pozitivne rezultate, ali ne i lažno negativne. Ukoliko pomoću *bloom filtera* nije pronađen sažetak virusa, sigurno neće biti pronađen ni nekom standardnom usporedbom.

Odabir algoritma kojim će se izvršiti točno prepoznavanje je proizvoljan, primjerice *Boyer-Moore* ili korištenje sekundarne *hash* tablice. Uspješnost ovakvog ubrzanja ovisi o odabiru *hash* funkcija i o parametrima **k**, **β**, **N** [13].

6. Zaključak

U ovom radu opisani su temeljni principi na kojima rade zlonamjerni programi. Razumijevanje tih principa potrebno je za stvaranje uspješnih metoda obrane, nužnih za održavanje sigurnog rada računalnih sustava. Prednost autora zlonamjernih programa stvara stalnu utrku za novim pristupima obrane.

U praktičnom dijelu rada, program *PEscanner* prikazuje implementaciju jednostavnog programa za otkrivanje zlonamjernih programa u izvršnim datotekama na *Win32* operacijskim sustavima. *PEscanner* rastavlja i pretražuje strukturu *PE* izvršnih datoteka. Postoji velik broj mogućnosti nadogradnje za takav program, koje su nužne za stvaranje suvremenog *anti-malware* alata. Navedeni program se bavi otkrivanjem potpisa *malware-a* i otkrivanjem sumnjivih heurističkih pokazatelja na *PE* datotekama i procesima na *Win32* operacijskom sustavu. Taj pristup predstavlja samo mali dio obrane protiv zlonamjernih programa.

7. Literatura

[1] Fred Cohen, „Computer Viruses: Theory and Experiments", Computers and Security 6, Elsevier Advanced Technology Publications, 1987

[2] Peter Szor, The Art of Computer Virus Research and Defense, Addison Wesley, 2005

[3] David M. Chess and Steve R. White, An Undetectable Computer Virus, IBM Thomas J. Watson Research Center na Internet adresi
http://www.research.ibm.com/antivirus/SciPapers/VB2000DC.htm

[4] Matt Pietrek, An In-Depth Look into the Win32 Portable Executable File Format, na Internet adresi
http://msdn.microsoft.com/msdnmag/issues/02/02/PE/

[5] Opis funkcija MSSQL servera na Internet adresi
http://msdn2.microsoft.com/en-us/library/ms175046.aspx

[6] Linux man stranice

[7] Stranica o regularnim izrazima na Internet adresi http://www.regular-expressions.info/

[8] Mihai Christodorescu, Somesh Jha, „Static Analysis of Executables to Detect Malicious Patterns", Computer Sciences Department, University of Wisconsin, Madison, na Internet adresi
http://www.cs.wisc.edu/wisa/papers/security03/cj03.html

[9] Skripta iz kolegija Operacijski sustavi 2, Fakultet elektrotehnike i računarstva, Zagreb

[10] *Game of life* Johna Conwaya na Internet adresi http://www.bitstorm.org/gameoflife/

[11] Randy Kath, The Virtual-Memory Manager in Windows NT, na Internet adresi
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dngenlib/html/msdn_ntvmm.asp

[12] MSDN Library, na Internet adresi http://msdn.microsoft.com/library/

[13] Ozgun Erdogan and Pei Cao, Hash-AV Fast Virus Signature Scanning by Cache-Resident Filters, Department of Computer Science, Stanford University

[14] CARO (Computer Antivirus Research Organization) na Internet adresi
http://www.caro.org/tiki-index.php?page=CaroNamingScheme

[15] Peter Szor and Peter Ferrie, Hunting for the Metamorphic, Symantec Corporation, na Internet adresi http://www.symantec.com/avcenter/reference/hunting.for.metamorphic.pdf

[16] Mihai Christodorescu, Detecting Malicious Patterns in Executables via Model Checking, University of Wisconsin, Madison, na Internet adresi http://www.cs.wisc.edu/wisa/presentations/2002/0712/mihai/mihai.7.12.02.pdf

[17] Sandeep Kumar and Eugene H. Spafford, A Generic Virus Scanner in C++, Department of Computer SciencesPurdue University, na Internet adresi http://vx.netlux.org/lib/aes04.html

[18] Igor Muttik, Stripping Down an AV Engine, Network Associates Inc (McAfee Division), na Internet adresi http://www.mcafee.com/us/local_content/white_papers/threat_center/wp_imuttik_vb_conf_2000.pdf

[19] Opis logičkih bombi, na Internet adresi http://www.tech-faq.com/logic-bomb.shtml

[20] Greg Hoglund and James Butler, Rootkits: Subverting the Windows Kernel, Addison Wesley Professional, 2005

[21] DoS napadi, na Internet adresi http://www.cert.org/tech_tips/denial_of_service.html

[22] Clam Antivirus, na Internet adresi http://www.clamav.net/

[23] Opis snage lozinki, na Internet adresi http://www.microsoft.com/athome/security/privacy/password.mspx

[24] Boyer-Moore algoritam, na Internet adresi http://www.inf.fh-flensburg.de/lang/algorithmen/pattern/bmen.htm

8. Dodaci

Dodatak 1: Tablica standardnih platformi koju specificira organizacija *CARO* [14]

Kratki oblik	Dugi oblik	Komentari
ABAP	ABAP	Malware za SAP /R3 Advanced Business Application Programming okruženje.
AmigaOS	AmigaOS	Malware za AmigaOS.
Apple2	AppleII	Malware zaAppleII.
BeOS	BeOS	Malware za BeOS.
Boot	Boot	Zahtijeva *MBR* i/ili disk kompatibilan sa *IBM PC system boot sector*-om. Rijetko se koristi.
DOS	DOS	*Malware* u *DOS COM, EXE (MZ)* ili *SYS* formatu zahtjevaju neku verziju *MS-DOS-a* ili blisko kompatibilnog operacijskog sustava. *Malware* koji radi samo pod specifičnom verzijom *MS-DOS-a* nije predviđen za ovu platformu.
EPOC	EPOC	Zahtijeva *EPOC OS*
Java	Java	*Malware* zahtijeva *Java runtime environment (JRE)*
MacOS	MacOS	Zahtijeva *Macintosh OS* prije *OS X.*
MeOS	MenuetOS	Zahtijeva *MenuetOS*.
MSIL	MSIL	Zahtijeva *Microsoft Intermediate Language* interpretersku platformu.
Mul	Multi	Ovo je pseudo-platforma koja se koristi za neke specijalne slučajeve.
PalmOS	PalmOS	Zahtijeva verziju *PalmOS*-a.
OS2	OS2	Zahtijeva *OS/2.*
OSX	OSX	Zahtijeva *Macintosh OS X* ili neku noviju sličnu verziju.
SymbOS	SymbianOS	Zahtijeva *Symbian OS 6.0* ili noviju verziju.
W16	Win16	Zahtijeva nekiod 16-bitnih *Windows x86* operacijskih sustava.
W32	Win32	Zahtijeva jednu od 32-bitnih *Windows x86* operacijskih sustava..
W64	Win64	Zahtijeva jednu od 64-bitnih *Windows x86* operacijskih sustava.. Nema razlike između IA64, AMD64 ili nekog budućeg *CPU.*
WCE	WinCE	Zahtijeva operacijski sustav iz *Windows CE* porodice platfomi.
WM	WordMacro	Makro *malware* za *WordBasic* uključen u *WinWord 6.0, Word 95* i *Word for Mac 5.x.*
W2M	Word2Macro	Makro *malware* za *WordBasic* uključen u *WinWord 2.0.*
W97M	Word97Macro	Makro *malware* za *Visual Basic for Applications (VBA) v5.0 for Word.*
AM	AccessMacro	Makro *malware* za *AccessBasic*.
A97M	Access97Macro	Makro *malware* za *Visual Basic for Applications (VBA) v5.0 for Access.*
P97M	Project97Macro	Makro *malware* za *Visual Basic for Applications*

		(VBA) v5.0 for Project., that shipped in Project 97 and later. As for W97M, changes in VBA versions between Project 97 and 2003 inclusive are insufficient to justify distinguishing the platforms.
PP97M	PowerPoint97Macro	Macro *malware* for *Visual Basic for Applications (VBA) v5.0 for PowerPoint.*
V5M	Visio5Macro	Macro *malware* for *Visual Basic for Applications (VBA) v5.0 for Visio.*
XF	ExcelFormula	*Malware* baziran na *Excel Formula language* unutar Excel alata.
XM	ExcelMacro	Makro *malware* za *Visual Basic for Applications (VBA) v3.0* za *Excel.*
X97M	Excel97Macro	Makro *malware* za *Visual Basic for Applications (VBA) v5.0 for Excel.*
O97M	Office97Macro	Ovo je pseudo platforma rezervirana za makro *malware* koji se inficira preko bar dvije aplikacije unutar *Office 97 ili novijih verzija.*
ACM	AutoCadMacro	*VBA v5.0* makro *malware* za *AutoCAD r11* i novije verzije.
ActnS	ActionScript	Zahtijeva *Macromedia ActionScript* interpreter koji se nalazi u nekim aplikacijama koje koriste *ShockWave Flash.*
LM	LotusMacro	*Malware* za *Lotus 1-2-3* makro okruženje.
WPM	WordProMacro	*Malware* za *Lotus WordPro* makro okruženje.
AplS	AppleScript	Zahtijeva *AppleScript* interpreter.
APM	AmiProMacro	Makro *malware* za *AmiPro.*
AutoLISP	AutoLISPScript	Zahtijeva *AutoLISP* interpreter poput onog uključenog u *AutoCAD..*
BAT	BAT	*Malware* koji zahtijeva *DOS, Windows ili NT* komandni interpreter.
CSC	CorelScript	*Malware* koji zahtijeva *CorelScript* interpreter.
DCL	DCLScript	*Malware* koji zahtijeva *Digital Command Language* interpreter.
HLP	WinHelpScript	Zahtijeva interpreter za *WinHelp display engine.*
INF	INFScript	Zahtijeva jedan od *Windows INF (installer)* interpretera
JS	JScript, JavaScript	Zahtijeva *JScript* i/ili *JavaScript* interpreter.
mIRC	mIRCScript	Zahtijeva *mIRC* interpreter.
MPB	MapBasic	Zahtijeva *MapBasic* okruženje *MapInfo* alata.
Perl	Perl	Zahtijeva *Perl* interpreter.
PHP	PHPScript	Zahtijeva *PHP* interpreter.
Pirch	PirchScript	Zahtijeva *Pirch (Windows IRC klijent)* interpreter.
PS	PostScript	Zahtijeva *PostScript* interpreter.
Py	Python	Zahtijeva *Python* interpreter.
REG	Registry	Zahtijeva interpreter *Windows registry* datoteka (.REG) .
Ruby	RubyScript	Zahtijeva *Ruby* interpreter.
SH	ShellScript	Zahtijeva *Unix shell* (ili srodni) interpreter. Za *shell malware* specifičan za *Linux, Solaris, HP-UX*

VBS	VBScript, VisualBasicScript	Zahtijeva *VBS* interpreter.
WBS	WinBATScript	*Malware* koji zahtijeva *Wilson WindowWare WinBatch* interpreter.
Unix	Unix	Ovo je preferirana platforma za *binary malware* za *Unix(-oidne)* platforme. Dok ne poraste broj *malware-a* za platforme bazirane na *Unix-u* nema potrebe za specifičnije određivanje platforme.
BSD	BSD	Za *malware* specifičan za *BSD* bazirane platforme osim za *OS X*. Unix je ipak preferirano ime.
Linux	Linux	Za *malware* specifičan za *Linux* platforme. Unix je ipak preferirano ime.
Solaris	Solaris	Za *malware* specifičan za *Solaris* platforme. Unix je ipak preferirano ime.