VILNIAUS GEDIMINO TECHNIKOS UNIVERSITETAS

FUNDAMENTINIŲ MOKSLŲ FAKULTETAS

INFORMACINIŲ SISTEMŲ KATEDRA

Dainius Čeponis

WINDOWS API FUNKCIJŲ PANAUDOJIMAS REALIOJO LAIKO NUSKAITYMO SISTEMOSE

WINDOWS API FUNCTIONS USAGE IN A REAL-TIME SCANNING SYSTEMS

Baigiamasis magistro darbas

Inžinerinės informatikos studijų programa, valstybinis kodas 62409P11

Informacinių sistemų specializacija

Informatikos mokslo kryptis

Vilnius, 2010

VILNIAUS GEDIMINO TECHNIKOS UNIVERSITETAS
FUNDAMENTINIŲ MOKSLŲ FAKULTETAS
INFORMACINIŲ SISTEMŲ KATEDRA

TVIRTINU

Katedros vedėjas

_____
(Parašas)

_____
(Vardas, pavardė)

_____
(Data)

Dainius Čeponis

WINDOWS API FUNKCIJŲ PANAUDOJIMAS REALIOJO LAIKO NUSKAITYMO SISTEMOSE

WINDOWS API FUNCTIONS USAGE IN A REAL-TIME SCANNING SYSTEMS

Baigiamasis magistro darbas

Inžinerinės informatikos studijų programa, valstybinis kodas 62409P11
Informacinių sistemų specializacija
Informatikos mokslo kryptis

Vadovas_____ _____ _____
(Moksl. laipsnis, vardas, pavardė) (Parašas) (Data)

Konsultantas_____ _____ _____
(Moksl. laipsnis, vardas, pavardė) (Parašas) (Data)

Konsultantas_____ _____ _____
(Moksl. laipsnis, vardas, pavardė) (Parašas) (Data)

Vilnius, 2010

Vilniaus Gedimino technikos universitetas
Fundamentinių mokslų fakultetas
Informacinių sistemų katedra

ISBN ISSN
Egz. sk. ……..
Data ….-….-….

Inžinerinės informatikos studijų programos baigiamasis magistro darbas
Pavadinimas **Windows API funkcijų panaudojimas realiojo laiko nuskaitymo sistemose**
Autorius **Dainius Čeponis** Vadovas doc. dr. **Antanas Čenys**

Kalba
☐ lietuvių
☐ užsienio

Anotacija

Darbe aptariamos virusų aptikimo metodikos, jų raida. Pristatomi nauji virusų aptikimo metodai ir genetinių algoritmų pritaikymas juose. Pirmoje praktinėje dalyje atliktas Windows API funkcijų perėmimo bibliotekų palyginimas.

Antroje praktinėje dalyje parašyta realiojo laiko nuskaitymo programa. Programa veikia naudodama nemokamas ClamAV virusų parašų duomenų bazes. Atliktas programos palyginimas su kitomis nemokamomis antivirusinėmis programomis.

Prasminiai žodžiai: Windows API, funkcijų perėmimas, realiojo laiko nuskaitymas, kenkimo programinis kodas, Detours, EasyHook, ClamAV.

Vilnius Gediminas Technical University
Fundamental science faculty
Information systems department

ISBN ISSN
Copies No. ……
Date ….-….-….

Engineering informatics study programme master thesis.

Title: **Windows API functions usage in real-time scanning systems**

Author **Dainius Čeponis** Academic supervisor **Antanas Čenys**

Thesis language

☐ Lithuanian

☐ Foreign (English)

Annotation

In work are described virus scanning techniques, they historical appearance. Also presented new scanning techniques, including genetic algorithms usage. In first practical part presented Windows API hooking libraries, they tests.

In second practical part created real-time system scanning program. ClamAV free databases are used for files checking. Program tested with others free antivirus solutions.

Keywords: Windows API, hooking, real-time scanning, virus, Detours, EasyHook, ClamAV.

TURINYS

PAVEIKSLŲ SĄRAŠAS

LENTELIŲ SĄRAŠAS

ĮVADAS

Sparčiai tobulėjant kompiuterinėms technologijoms, tobulėja ir virusų kūrėjai. Neretai virusai kuriami norint pasipelnyti, o ne tik parodyti savo sugebėjimus. Taikomasi ne tik į namų vartotojus, bet ir į įmonių tinklus. Todėl būtina kurti naujus bei tobulinti senus įrankius virusų aptikimui ir apsaugai nuo jų. Pats efektyviausias įrankis – antivirusinė programa. Dažniausiai ji veikia realiu laiku ir stebi vartotojo bei sistemos veiksmus. Aptikus įtartiną rinkmeną arba taikomosios aplikacijos veikimą, ji praneša vartotojui arba pati atlieka viruso pašalinimą.

Antivirusinių programų rinkmenų nuskaitymo metodai susideda iš kelių pagrindinių metodų. Tai paprastas rinkmenos nuskaitymas ir palyginimas su virusų duomenų baze, rinkmenoje naudojamų komandų sekų palyginimas, įtartinos programos vykdymas virtualioje sistemoje. Padėti dar labiau sunkina tai, kad dauguma šiuo metu egzistuojančių virusų sugeba keisti savo veikimo principus, taip apgaudami antivirusines programas.

Realiojo laiko Windows sistemos stebėjimas – sudėtingas darbas. Tai reikia atlikti greitai ir efektyviai, nepakenkiant pačiai sistemai. Antivirusinių programų autoriai sistemos stebėjimo metodais nesidalina, bet vienas iš jau seniai žinomų yra stebėjimas perimant sistemoje atliekamus Windows API (*application programming interface*) funkcijų kvietimus. Tokiu būdu, perimant darbo su rinkmenomis funkcijas, galima žinoti kada jos atidaromos arba nuskaitomos.

Šio darbo tikslas – apžvelgti virusų aptikimo metodikas, jų raidą. Aptarti Windows API funkcijas tinkamas realiojo laiko rinkmenų sistemos stebėjimui. Aptarti trečiųjų šalių pateikiamas bibliotekas skirtas Windows API funkcijų perėmimui ir atlikti jų palyginimą. Surinktas žinias realizuoti realaus laiko nuskaitymo sistemoje.

Darbo uždaviniai yra šie:

- Aptarti virusų aptikimo esamas metodikas. Pristatyti šiuo metu siūlomas naujas metodikas.
- Aptarti Windows API funkcijas, skirtas darbui su rinkmenomis.
- Palyginti SetWindowsHookEx(), Detours ir EasyHook suteikiamas galimybes. Pirmoje praktinėje dalyje atlikti jų spartos testus.
- Antroje praktinėje dalyje sukurti taikomąją programą, kuri realiu laiku atlieka sistemos stebėjimą ir nuskaito paliestas rinkmenas. Taip pat programa turi turėti standartines antivirusinių programų funkcijas: operatyviosios atminties, rinkmenų, katalogų nuskaitymas pagal pareikalavimą bei virusų duomenų bazės atnaujinimas. Atlikti taikomosios aplikacijos testus.
- Pateikti darbo išvadas ir pasiūlymus tolesniam tyrimui šia kryptimi.

Šio *darbo struktūra* padalinta į penkias dalis. Pirmame, antrame, trečiame ir ketvirtame skyriuje pateikiama analitinė-metodinė dalis, nagrinėjama teorinė medžiaga. Ketvirtojo skyriaus

pabaigoje atlikta pirmoji praktinė dalis. Penktajame skyriuje pateikiama pagrindinė, antroji, praktinė dalis. Joje aptariamas realiojo laiko nuskaitymo programos veikimas, įgyvendinimas bei pateikiami testo rezultatai.

1. Kenkimo programinė įranga

Virusai arba kenkimo programinė įranga kuriama norint pakenkti arba pasisavinti vartotojo duomenis jam nežinant. Dažniausiai jos atliekami veiksmai: kompiuteryje esančios informacijos sugadinimas, atakos prieš kitus kompiuterius, kompiuterinių tinklų apkrovimo didinimas siekiant perkrovų, kompiuterio valdymo perėmimas[10].

Pagrindiniai virusų tipai:

- **Kirminai** (angl. worms) yra virusų, kurie patys sugeba daugintis, atmaina. Didžiausias jų keliamas pavojus yra sugebėjimas daugintis dideliais kiekiais.
- **Trojanai** (dar kitaip vadinami Trojos arkliais) – tai kompiuterinės programos, besislepiančios kitose programose ir išoriškai atrodančios kaip naudingos, tačiau realiai sukeliančios kenksmingus padarinius.
- **Makrovirusai** – tai virusų rūšis, kuri pažeidžia dokumentus, sukurtus su taikomosiomis programomis, turinčiomis makrokomandų vykdymo priemonių. Dokumentas užkrečiamas, kai jis atidaromas programos lange, jei nėra uždraustas makrokomandų vykdymas.
- **Įkrovos** virusai – vieni pirmųjų kompiuterinių virusų. Tokie virusai plito iš diskelių ir veikdavo per kompiuterio DOS (angl. Disk Operating System) sistemą. Tokių virusų plitimo mastai nebuvo dideli ir šiuo metu praktikoje jų beveik nepasitaiko.
- **Polimorfiniai** virusai (angl. polymorphic virus). Šie virusai kiekvieną sykį užkrėsdami sistemą koduojami ir dekoduojami naudojant skirtingus algoritmus ir (de)kodavimo raktus. Toks maskavimo būdas labai apsunkina antivirusinių programų paieškas, nes praktiškai tampa neįmanoma aptikti virusų duomenų eilutes ar jų „parašus“.

Dar sunkiau aptinkami yra **metamorfiniai** virusai (angl. metamorphic viruses), kurie yra visiškai perrašomi po kiekvienos atakos. Tam jiems reikalingas metamorfinis variklis (vieno žinomiausių šių virusų W32/Simile variklis užėmė apie 90% viruso dydžio). Šie virusai dažniausiai yra kompleksiniai ir užima daug vietos[10].

Virusai pagal kenksmingumo lygį skirstomi į:

- **Nepavojingus virusus**, kurie kompiuterio veikimui nesukelia esminės žalos. Tokie virusai patys daugindamiesi užima tam tikrą kompiuterio atminties dalį, be to, jie gali patys pateikti į kompiuterio ekraną grafinius vaizdus, pranešimus, imituoti įvairius garsus ir pan[14].
- **Pavojingus virusus**, kurie gali sutrikdyti kompiuterio darbą. Tokie virusai sistemoje veikia vartotojui nematant ir iš pradžių nesukelia įtarimų, tačiau vėliau gali atsirasti įvairių sistemos darbo sutrikimų: atskirų programų ar kompiuterio procesoriaus darbo sulėtėjimas, nenumatytų programose simbolių atsiradimas ir t. t[14].

- **Labai pavojingus** virusus, kurie naikina kitas programas ir kompiuteryje esančius duomenis, vagia vertingą informaciją, ištrina būtiną sisteminę kompiuterio informaciją (pvz., rinkmenų išdėstymo lenteles). Taip pat virusas gali tapti labai pavojingas, kai nekontroliuojamai dauginasi plisdamas kompiuterių tinklais taip, kad sutrikdo viso tinklo darbą[14].

2. Virusų aptikimo metodikos

Apžvelgsime virusų aptikimo metodus, jų raidą. Skyriaus pabaigoje aptarsime šiandien aktualius metodus, kurie pakeičia senuosius. Visos antivirusinės programos veikia naudodamos virusų parašų duomenų bazes, tad pirmiausia aptarsime viruso parašo sąvoką[12].

2.1. Viruso parašo apibrėžimas

Antivirusų pasaulyje, parašas yra algoritmas arba skaitinė funkcija (skaičius, gautas iš teksto eilutės), kuri unikaliai identifikuoja tam tikrą virusą. Priklausomi nuo skaitytuvo tipo tai gali būti statinė skaitinė funkcija, kuri paprasčiausiai apskaičiuota pagal unikalaus viruso kodo fragmentą. Tai gali būti ir elgsena paremtas algoritmas, pvz. jei programa daro veiksmus X, Y, Z jis pažymimas kaip įtartinas ir apie tai pranešama vartotojui.

Vienas parašas gali atitikti keletą virusų. Todėl skaitytuvas gali aptikti naujus virusus, net ir nepažymėtus duomenų bazėje. Tai atliekama naudojant euristinį arba paprastąjį nuskaitymą[22]. Paprastasis nuskaitymas duoda mažiausiai naudos ieškant naujų virusų iš jau žinomų virusų šeimos (virusų tipas, kuris elgiasi panašiai arba net naudojasi tuo pačiu programiniu kodu). Galimybė aptikti naujus virusus euristiškai arba įprastai yra labai reikšmingas, nes nauji virusai atsiranda pavydėtinai sparčiu tempu.

2.2. Parašų atnaujinimas

Kiekvieną kartą, kai atrandamas naujas virusas, kuris neaptinkamas pagal turimas duomenų bazes, arba aptinkamas, bet neįmanoma jo tinkamai pašalinti dėl specifinės elgsenos, turi būti sukuriamas naujas viruso parašas. Po šito veiksmo naujas parašas ištestuojamas su antivirusine programa. Tuomet atnaujinta duomenų bazė gali būti prieinama antivirusinės programos naudotojams. Šie atnaujinimai suteikia galimybę tinkamai aptikti ir pašalinti naują virusą iš sistemos. Kai kuriais atvejais parašai gali būti panaikinami arba pakeičiami naujais, siekiant geriausių aptikimo ir pašalinimo savybių [30].

Priklausomai nuo skaitytuvo gamintojo, atnaujinama gali būti kas valandą, dieną arba netgi kas savaitę. Tai priklauso nuo to, kokio tipo yra programinė įranga, pvz. šnipinėjimo programų skaitytuvui užtenka ir kas savaitinio atnaujinimo, o virusų skaitytuvui ilgiausias periodas turi būti viena para. Žinoma nėra praktiška atnaujinti kai tik atsiranda koks naujas virusas. Tiekėjai paprastai sukaupia tam tikrą skaičių aprašų per tam tikrą periodą ir tik tuomet leidžia klientams parsisiųsti atnaujintą duomenų bazę[30].

Norint pasiekti didžiausią savo antivirusinės programos efektyvumą, reikia susikonfigūruoti, kad ji virusų duomenų bazės atnaujinimus parsisiųstų kaip galima dažniau.

2.3. Raida

Nuskaitymas pagal viruso parašą – pirmas metodas, kurio griebėsi antivirusinių programų kūrėjai. Vėliau šis metodas vis keitėsi, stiprėjo sujungdamas naujus patentus ir metodus. Besikeičianti kompiuterinė technika leido kurti vis sudėtingesnius, bet efektyviau veikiančius algoritmus. Čia apžvelgiama nuskaitymo metodų raida.

2.3.1. Teksto eilutės nuskaitymas

Tai pats paprasčiausias metodas, aptikti kompiuteriniam virusui. Jis naudoja tik virusui tipišką baitų (teksto) seką, kuri negali būti atrasta eilinėje programoje. Šios sekos saugomos duomenų bazėje, kurią naudoja antivirusinės programos. Virusui aptikti skiriamas tam tikras laikas, per kurį failas turi būti patikrinamas. Skiriamas laikas yra vienas iš didžiausią iššūkį keliančių veiksnių (dažniausiai tas laikas yra viena ar dvi sekundės). Nuskaitymo algoritmas šį laiką turi panaudoti labai protingai, kad sugebėtų per jį aptikti virusą.

Panagrinėkime gerai žinomą, bet jau labai seną virusą Stoned. Jis sukurtas 1987 m., Naujoje Zelandijoje. Tai buvo vienas pirmųjų virusų ir turėjo daugybę variantų. Apkrėstas šiuo virusu kompiuteris, su tikimybe 1/8 ekrane parašydavo „Your PC is now Stoned!“. Ši frazė būdavo išsaugojama standžiojo disko ir diskelio krovimosi sektoriuose kartu su tekstu „Legalise Marijuana“ [13].

```
seg000:7C40 BE 04 00                    mov     si, 4           ; Try it 4 times
seg000:7C40                             ;
seg000:7C43
seg000:7C43                 next:                               ; CODE XREF: sub_7C3A+27↓j
seg000:7C43 B8 01 02                    mov     ax, 201h        ; read one sector
seg000:7C46 0E                          push    cs
seg000:7C47 07                          pop     es
seg000:7C48                             assume es:seg000
seg000:7C48 BB 00 02                    mov     bx, 200h        ; to here
seg000:7C4B 33 C9                       xor     cx, cx
seg000:7C4D 8B D1                       mov     dx, cx
seg000:7C4F 41                          inc     cx
seg000:7C50 9C                          pushf
seg000:7C51 2E FF 1E 09 00              call    dword ptr cs:9  ; int 13
seg000:7C56 73 0E                       jnb     short fine
seg000:7C58 33 C0                       xor     ax, ax
seg000:7C5A 9C                          pushf
seg000:7C5B 2E FF 1E 09 00              call    dword ptr cs:9  ; int 13
seg000:7C60 4E                          dec     si
seg000:7C61 75 E0                       jnz     short next
seg000:7C63 EB 35                       jmp     short giveup
```

1 pav. Stoned viruso programinis kodas

1 pav. pavaizduotas Stoned viruso kodas. Tai yra tipinis jo asemblerio kodas. Pirmausia yra keturi bandymai nuskaityti pirmąjį disko sektorių. Tada į jį įrašyti jau paruoštus duomenis. Iš šio kodo išrinkti 16 baitų, kurie buvo paskelbti *Virus Bulletin* žurnale kaip viruso atpažinimo kodas:

0400 B801 020E 07BB 0002 33C9 8BD1 419C.

Tais laikais 16 baitų tekstas buvo pakankamo ilgio norint aptikti 16 bitų kenkimo kodą.

Paminėta teksto eilutė gali pasitaikyti ir Stoned viruso kopijose. Taip buvo įmanoma aptikti A, B ir C jo variantus. Tai netgi gali padėti aptikti artimai susijusius virusus, kurie priklauso skirtingoms šeimoms. Tai yra gerai, kadangi skaitytuvas, naudodamas šią teksto eilutę, gali aptikti daugiau virusų. Kitą vertus, tai gali būti blogai, kai visiškai nepanašus virusas gali būti aptiktas kaip Stoned. Tai gali suklaidinti vartotoją, ir jis galvos, jog aptiktas pakankamai nepiktas virusas, nors iš tikrųjų yra priešingai [30].

Aptikimo nesusipratimas gali kainuoti labai brangiai. Ypač jei antivirusinė programa bandys pašalinti klaidingai aptiktą virusą. Tokiu būdu ji gali pažeisti vartotojo duomenis.

2.3.2. Šablonai

Viruso unikalus tekstas gali būti saugomas kaip šablonas. Tai leidžia ignoruoti baitų sekas, ir naudoti reguliarias išraiškas. Štai Stoned viruso šablonas [12]:

0400 B801 020E 07BB ??02 %3 33C9 8BD1 419C.

Pateiktas šablonas bus interpretuojamas tokia tvarka:

1. *Sulyginame su 04 ir jei tinka, ieškome toliau.*
2. *Sulyginame su 00 ir jei tinka, ieškome toliau.*
3. *Sulyginame su B8 ir jei tinka, ieškome toliau.*

4. *Sulyginame su 01 ir jei tinka, ieškome toliau.*
5. *Sulyginame su 01 ir jei tinka, ieškome toliau.*
6. *Sulyginame su 0E ir jei tinka, ieškome toliau.*
7. *Sulyginame su 07 ir jei tinka, ieškome toliau.*
8. *Sulyginame su BB ir jei tinka, ieškome toliau.*
9. *Ignoruojame šį baitą.*
10. *Sulyginame su 02 ir jei tinka, ieškome toliau.*
11. *Sulyginame su 33 (šablonas nurodo, kad šalia 3 gali būti bet koks simbolis) ir jei tinka, ieškome toliau.*
12. *Sulyginame su C9 ir jei tinka, ieškome toliau.*
13. *Sulyginame su 8B ir jei tinka, ieškome toliau.*
14. *Sulyginame su D1 ir jei tinka, ieškome toliau.*
15. *Sulyginame su 41 ir jei tinka, ieškome toliau.*
16. *Sulyginame su 9C ir jei tinka, ieškome toliau.*

Šablonai leidžia naudoti pusbaičius, kurie įgalina labai preciziškai aptikti instrukcijų grupes. Taip gali būti aptinkami net kai kurie pirmieji užšifruoti (angl. encrypted) arba netgi polimorfiai virusai.

2.3.3. Neatitiktys

Neatitikimą teksto eilutėse išrado IBM Antivirus. Neatitiktys (angl. mismatches) leidžia *N* kiekį baitų tekste turėti bet kokią reikšmę, nepaisant pozicijos. Pavyzdžiui, tekstas *01 02 03 04 05 07 08 09* neatitikties reikšmė 2 atitiks visas kitas eilutes:

01 02 ***AA*** *04 05 06* ***BB*** *08 09 0A 0B 0C 0D 0E 0F 10*;

01 02 03 ***CC DD*** *06 07 08 09 0A 0B 0C 0D 0E 0F 10*;

01 ***EE*** *03 04 05 06 07* ***FF*** *09 0A 0B 0C 0D 0E 0F 10*;

Neatitiktys ypač naudingos kuriant aptikimo eilutes virusų šeimoms. Blogoji šio algoritmo pusė – tai gana lėta nuskaitymo technika [12].

2.3.4. Pradžios ir pabaigos nuskaitymas

Pradžia ir pabaiga nuskaitoma norint paspartinti viruso paiešką tikrinant tik failo pradžią ir pabaigą užuot analizavus visą jo turinį. Tokiu būdu nuskaitomi pirmi ir paskutiniai 2, 4 arba netgi 8 failo baitai. Tai truputį geresnis sprendimas nei ankščiau naudoti metodai. Spartėjant procesoriams, algoritmo efektyvumas priklausydavo vien nuo skaitymo ir rašymo trukmės į atmintį.

Taigi programuotojai nusprendė mažinti skaitymo ir rašymo skaičių. Kadangi tais laikais dauguma virusų keisdavo užkrėstų failų pradžią arba pabaigą, šis nuskaitymo metodas greitai paplito [12].

2.3.5. Išmanusis nuskaitymas

Išmanusis nuskaitymas pristatytas tuomet, kai atsirado pirmieji virusų mutavimo požymiai. Programos, kurios tai atlikdavo veikė su viruso asemblerio kodu ir mėgindavo pridėti neesmines instrukcijas prie jau esamų. Dažniausiai tai būdavo nieko nedarančios NOP instrukcijos. Sukompiliuota programa jau skirdavosi nuo originalaus viruso, bet atlikdavo tą patį darbą.

Išmanusis nuskaitymas praleisdavo NOP instrukcijas kurdamas viruso parašą. Taip pat buvo pasirenkamos viruso vietos, kurios neturėjo jokių nuorodų į papildomus duomenis arba kitas funkcijas. Tokie metodai padidino tikimybę, kad bus aptikti artimai susiję virusai iš tos pačios šeimos.

Ši technika taip pat naudinga kovojant su virusais, kurie pateikiami kaip scenarijai arba makrokomandos. Kadangi šie virusai yra paprasti tekstiniai failai, tai jie gali būti lengvai pakeičiami pridėjus daug tuščių tarpų, eilučių arba TAB simbolių į kodą. Šie simboliai taip pat buvo neįtraukiami į viruso parašą, gaminamą išmaniajam skaitytuvui [12].

2.3.6. Skeleto panaudojimas

Skeleto panaudojimas išrastas rusų programuotojo Eugenijaus Kasperskio. Dabar jis yra Kasperskio antivirusinės programos vadovas. Šis metodas labai naudingas aptinkant makrokomandų virusus ir jų šeimas. Vietoje parinkdamas paprastą teksto eilutę ar jos skaitinę funkciją, skaitytuvas analizuoja makro funkcijas kiekvienoje failo eilutėje ir atmeta neesmines komandas, taip pat ir ankščiau minėtus nereikalingus simbolius. Rezultatas – makrofunkcijų rinkinys, pasirodantis tik makrovirusuose [30].

2.3.7. Beveik tikslus aptikimo būdas

Norint dar tiksliau aptikti virusus buvo pradėtas naudoti *beveik tikslus aptikimo būdas* (angl. nearly exact identification). Pavyzdžiui, vietoje vienos tekstinės eilutės buvo naudojamos dvi eilutės kiekvienam virusui. Skaitytuvas gali aptikti virusą pagal pirmąją eilutę ir atmesti išvalytą failą, motyvuodamas tuo, kad tai gali būti viruso atmaina, ir viruso pašalinimas gali būti neefektyvus. Kai tuo tarpu aptikus virusą pagal dvi teksto eilutes, net ir aptikus jo kloną, yra didesnė tikimybė, kad virusas bus panaikintas tinkamai.

Kitas tikslaus aptikimo metodo būdas yra panaudoti kontrolinės sumos patikrinimą (angl. checksum). Kontrolinė suma paskaičiuojama iš viruso unikalaus kodo. Tai leidžia sumažinti

virusų duomenų bazės dydį, nes tiek mažo, tiek didelio viruso kodo kontrolinė suma užima tokį patį kiekį baitų [4].

2.3.8. Tikslus aptikimo būdas

Tai vienintelis būdas, kuris garantuoja viruso kopijų aptikimą. Šis metodas sukombinuotas iš pirmos kartos teksto eilučių skaitytuvų. Bet skirtingai nei *beveik tikslus aptikimo būdas*, kai naudojama vienos viruso dalies kontrolinė suma, šis metodas naudoja unikalių viruso dalių kontrolines sumas. Visų pirma sudaromas tokių dalių žemėlapis, o tada jam suskaičiuojama kontrolinė suma.

Toliau pateikiamos kelios Stoned viruso kopijos ir jų žemėlapiai:

```
Viruso pavadinimas: Stoned.A
Viruso žemėlapis: 0x0-0x7 0xD-0xE 0x11-0x1B7
Kontrolinė suma: 0x3523D929

0000:0180 0333DBFEC1CD13EB C507596F75722050 ..........Your P
0000:0190 43206973206E6F77 2053746F6E656421 C is now Stoned!
0000:01A0 070D0A0A004C4547 414C495345204D41 .....LEGALISE MA
0000:01B0 52494A55414E4121 0000000000000000 RIJUANA!........

Viruso pavadinimas: Stoned.B
Viruso žemėlapis: 0x0-0x7 0xD-0xE 0x11-0x1B7
Kontrolinė suma: 0x3523C769
0000:0180 0333DBFEC1CD13EB C507596F75722050 ..........Your P
0000:0190 43206973206E6F77 2073746F6E656421 C is now stoned!
0000:01A0 070D0A0A004C4547 414C495A45004D41 .....LEGALIZE.MA
0000:01B0 52494A55414E4121 0000000000000000 RIJUANA!........
```

Kaip matyti, viruso fragmentai beveik nesiskiria. Bet B kopijoje autorius pakeitė tik kelis teksto baitus, todėl kontrolinė suma skiriasi nuo A viruso varianto [30].

2.3.9. Algoritminis nuskaitymas

Šio metodo pavadinimas yra šiek tiek klaidinantis. Kai standartiniai nuskaitymo metodai nebegalėjo susitvarkyti, su virusais, reikėjo naujo specifinio metodo. Jis buvo pavadintas

algoritminiu nuskaitymu, bet *specifinis virusų aptikimo algoritmas* būtu tikslesnis pavadinimas. Pirmosios metodo versijos buvo paprastos, sudarytos iš paprastų funkcijų, kurios buvo tiesiog įdedamos į pagrindinį nuskaitymo modulį.

Deja tokios funkcijos pridarydavo daug problemų. Pirma tas aptikimo funkcijas būdavo sunku perkelti į kitą sistemą. Antra, atsirado stabilumo problema, kai sutrikus šių funkcijų darbui, sutrikdavo ir pagrindinio modulio darbas.

Šių problemų sprendimas buvo virusų skenavimo kalba. Ji leido labai paprastai išnagrinėti failą, ieškant virusų teksto eilučių.

Vėliau še algoritmai buvo parašyti kalba, panašia į Java. Tai leido juos perkelti į skirtingas platformas ir vykdyti virtualiose mašinose. Tai reiškia, kad algoritmas veiks ir PC kompiuteryje ir IBM mašinoje. Šio nešiojamojo metodo minusas – pakankamai lėtas vykdymas.

Ateityje tikimasi, kad algoritmo skaitytuvai jau turės JIT (Just In Time) sistemą. Tai leis realiu laiku kompiliuoti algoritmą. Tai labai panašu į Microsoft Windows naudojama .NET technologiją. Pavyzdžiu, jei algoritmas vykdomas Intel platformoje, tai jis realiu laiku ir sukompiliuojamas tai sistemai [30].

3. Nauji virusų aptikimo metodai

3.1. Dinaminiai virusų aptikimo metodai

Dinaminis aptikimo metodas vykdo programos kodą ir stebi jo veikimą. Programa ieško jai žinomų virusų veiklos požymių (taip pat ir mėginimų užkrėsti ir išvengti aptikimo). Į žinomus metodus įtraukti bandymai rašyti į įkrovos sektorius (angl. boot sector), keisti pertraukties vektorius (angl. interrupt vectors), perrašyti sisteminius failus ir kiti. Pavyzdžiui, daugumai virusų galiausiai reikia pasinaudoti tam tikru sistemos funkcionalumu – rašymu ir skaitymu iš tam tikro failo. Nesvarbu, kaip smarkiai jie paslėpti programiniame viruso kode, jie iškviečiami vykdant programa. Antivirusinė programa geriausiai atlieka savo darbą, kai normalus sistemos darbas labai skiriasi nuo užkrėstos. Galimas dinamis viruso parašas:

1. Atidaroma programa su skaitymo ir rašymo leidimais.
2. Nuskaitoma failo pradžia, kurioje yra programos pradžios adresas.
3. Į tą pačią vietą įrašoma kita informacija.
4. Nukeliaujama į failo pabaigą.
5. Failas uždaromas.

Ši metodą naudoja keletas skirtingų antivirusinių programų tipu. Vienas jų – elgsenos blokatorius (angl. behavior blocker) veikia realiame kompiuteryje, vykdo programas, o aptikęs

įtartinų veiksmų grandinę perspėja vartotoją. Vartotojas gali leisti arba neleisti toliau vykdyti instrukcijas (arba sprendimai priimami automatiškai). Priešingai elgiasi antivirusai, naudojantys emuliaciją. Jie programą įvykdo virtualioje aplinkoje ir praneša vartotojui apie pasekmes. Toks tikrinimas yra patikimesnis, nes nėra padaroma žala realiai vartotojo sistemai. Priklausomai nuo to, kaip antivirusai imituoja virtualią mašiną, virusas gali ir nesuprasti, kad yra vykdomas ne realioje aplinkoje[33].

Programinio kodo emuliacija yra patikimas metodas, ypač jei ieškoma koduotų (angl. encrypted) ir polimorfinių (angl. polymorphic) virusų[23]. Koduoti ir poliforminiai virusai atkoduoja save tuomet, kai jau patenka į kompiuterio atmintį. Jei emuliatorius veikia pakankamai ilgai, tai galu galiausiai atkoduotas programinis viruso kodas bus pastebėtas, o tada jį jau atpažins paprastas skaitytuvas, veikiantis pagal parašų atpažinimą[32]. Paprastas skaitytuvas atmintį patikrina tik tuo atveju, jei pasiekiamas maksimalus iteracijų skaičius (dekodavimo) arba kitomis aplinkybėmis. Žinoma, viruso parašo tikrinimas gali būti atliekamas ir periodiškai. Tokiu atveju visas viruso išsikodavimas nereikalingas, užtenka, kad programinis kodas pasiektų tam tikrą ilgį.

3.2. Euristinis virusų aptikimo metodas

Euristinės virusų aptikimo metodikos naudojamos aptikti naujiems, dar neužfiksuotiems virusams. Vienintelis jų didelis minusas – labai dažni netikri pavojaus paskelbimai (ang. false positives). Tai yra neefektyvu vartotojams, kurių darbas pertraukiamas atradus netikrą pavojų. Nepaisant to, euristiniai nuskaitymo metodai pasiteisina ir duoda daug naudos.

Euristiniai metodai dažniausiai naudojami naujoms jau žinomoms virusų šeimoms aptikti. Naudojami statiniai arba dinaminiai metodai. Statinė euristika paremta failo formatų ir jo kodo fragmentų analizavimu. Dinaminė euristika naudoja programinio kodo emuliaciją imituodama procesorių bei operacinę sistemą. Tuo metu ieškoma įtartinų operacijų, kurių kombinacija gali būti viruso veikimo požymis[30].

Toliau apžvelgsime keletą pagrindinių veiklų ir požymių, kurių ieško antivirusinė programa, naudodama euristinį nuskaitymo metodą. Dauguma jų paremti vykdomųjų programų struktūros tikrinimu. Įvairūs jų (programų) struktūros pakitimai ir nuokrypiai nuo standartinės gali būti laikomi viruso požymiu, jei tik yra labai įtartini[12].

- **Programinio kodo vykdymas prasideda paskutinėje dalyje**. Vykdomosios programos turi svarbų reikalavimą – skirtingos funkcinės dalys atskirtos į logines dalis. Dauguma virusų pakeičia programos vykdymo bloko vietą – perkelia ją į pačia pabaigą (ten kur lengva įterpti savo norimą funkcionalumą).

Naudojant standartinius kompiliatorius, taip nėra daroma, tad galima įtarti, kad kažkas buvo padaryta turint tam tikrų ketinimų.

- **Įtartinas programinio kodo vykdymo nukreipimas**. Kai kurie virusai nekeičia programos pradžios vietos. Vietoje to jie įterpia nukreipiančiąją (angl. jump) JMP komandą į savo funkcionalumą, kuris yra kitoje loginėje dalyje. Tai yra labai įtartina, kai iš pagrindinės programinio kodo vykdymo loginės dalies vyksta nukreipimas į kitą loginę dalį.
- **Įtartinas loginės dalies pavadinimas**. Įtarimą kelia, jei valdymą gauna loginės dalys, kurios paprastai neturi jokio vykdomojo kodo. Tokios dalys yra .reloc, .debug. Jei yra įvykdomas kodas, esantis šiose dalyse, turi būti pažymimas kaip įtartinas failas.
- **Netikslus antraštėje esantis programinio kodo dydis**. Dauguma virusų, įterpę kenksmingą kodą, nepakeičia SizeOfCode įrašo reikšmės. Tad, jei apskaičiuotas dydis skiriasi nuo reikalaujamo, yra galimybė, kad programa užkrėsta.

Euristiniuose nuskaitymo metoduose buvo panaudoti ir neuroniniai tinklai. Neuroniniai tinklai yra dirbtinio kompiuterio intelekto dalis, pasižyminti gebėjimu mokytis ir pritaikyti savo įgytas žinias.

IBM tyrėjai neuroninius tinklus panaudojo įkrovos virusų aptikimui. Buvo panaudota apie 50 virusų pavyzdžių. Po apmokymo neuroninis tinklas sugebėjo aptikti 85% įkrovos virusų, kai netikri pavojaus paskelbimai neviršijo 1%[30].

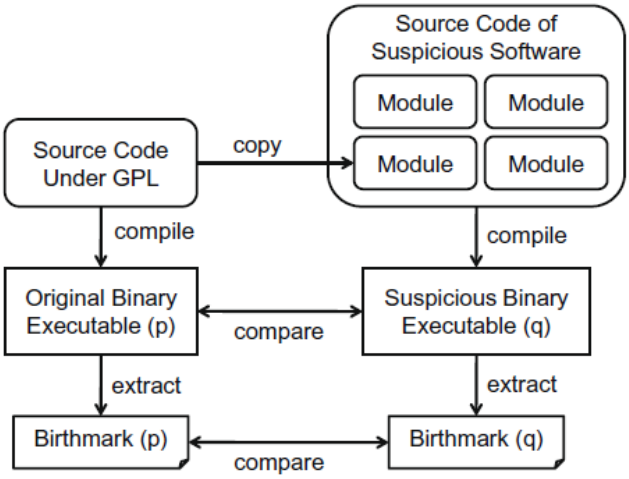
Neuroninių tinklų efektyvumas labai priklauso nuo apmokymo. Kuo tikslesni apmokymo duomenys pateikiami, tuo geresni virusų aptikimo rezultatai. Praktikoje neuroniniai tinklai efektyviai aptinka apmokymų metų naudotas virusų atmainas.

IBM sukurtas neuroninių tinklų variklis (angl. engine) buvo panaudotas ir Symantec kompanijos antivirusinėje programoje. Pastebėjus, kad šis metodas pateikia labai mažai netikrų pavojų paskelbimų, nuspręsta jį įdėti į standartinį skaitymo algoritmą (tai nebepriklausė nuo vartojo pasirinktų programos nuostatų)[30].

3.3. Vykdymo sekos kenkimo programiniam kodui aptikti

Egzistuoja Windows programų API antspaudai (angl. birthmark). Tai unikalios reikšmės, kurios gali būti naudojamos programai atpažinti[20]. Programų antspaudų sistemos sugeba sugeneruoti programos antspaudą ir apskaičiuoti panašumą su kitomis programomis. Dažniausiai jos pritaikomos ieškant programinio kodo vagysčių (labai populiaru universitetuose

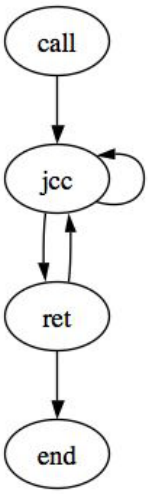
aptinkant darbų kopijas). 2 pav. pateikiamas tokių sistemų veikimo principas. Norint aptikti kopiją, apskaičiuojami abiejų programų antspaudai, jie palyginami. Kuo didesnis sutapimo laipsnis, tuo didesnė tikimybė, kad tikrinama programa yra kopija. Šis metodas taip pat naudojamas kenksmingam kodui atpažinti[35].



2 pav. Programų kopijų aptikimas[35]

Keturioliktame USENIX simpoziume buvo pasiūlytas sprendimas kaip apsisaugoti nuo nuotolinio Windows API funkcijų kvietimo[6]. Pasiūlyta saugoti funkcijų lenteles, kuriose būtu surašyti jų kvietimo adresai. Taip galima apsaugoti nuo lentelėje nesančių funkcijų kvietimo.

Egzistuoja ir kitų panašių virusų atpažinimo metodų. Vienas jų – programos srautų diagramos sudarymas[11]. Pagal tam tikras taisykles sudaromos programose naudojamų asemblerinių komandų srautų diagramos. Jos sudedamos į duomenų bazes ir vėliau lyginamos su pateiktos programos diagrama. 3 pav. pateikiamas tokio srauto pavyzdys.

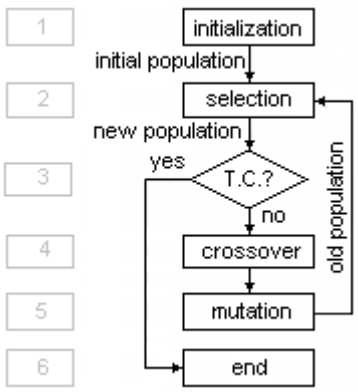


3 pav. Kenksmingos programos asemblerinių komandų srautas[11]

3.4. Genetinių algoritmų panaudojimas apsaugos sistemose

Genetiniai algoritmai (GA) – algoritmai, paremti biologijos žiniomis apie gyvybės evoliuciją. GA yra tam tikra evoliucinių algoritmų klasė, naudojanti gamtoje egzistuojančius gyvybės evoliucinius mechanizmus: paveldėjimą, mutaciją, natūraliąją atranką ir rekombinaciją. Genetiniai algoritmai yra metodas analizuoti duomenis, jų dėka galima rasti apytikslį užduoties sprendimą. Sprendimas randamas naudojant evoliucinį ciklą, veikiantį gamtoje. Genetiniai algoritmai tinka ne visur, tačiau jais galima rasti palyginti neblogus sprendinius užduočių, kurių tikslaus sprendimo algoritmai nežinomi.

GA veikimo schema pavaizduota 1 pav.



4 pav. GA veikimo schema[28]

Inicijavimo fazėje (1 pav., 1) generuojama pradinė atsitiktinių sprendimų populiacija. Kiekvienas sprendimas atvaizduojamas kaip iš genų sudaryta chromosoma. Atrankos fazėje (1 pav., 2) sprendimai atrenkami tinkamumo funkcijos pagalba ir, jei nėra pasiekiama nutraukimo sąlyga (1 pav., 3), pradeda veikti evoliuciniai sankirtos ir mutacijų mechanizmai (1 pav., 4,5). Tuo atveju, jei pasiekiama nutraukimo sąlyga, algoritmo vykdymas yra baigiamas (1 pav., 6) [28].

Daugelyje apsaugos sistemų GA algoritmai naudojami kartu su neuroniniais tinklais. Taip pasiekiamas maksimalus efektyvumas mokymosi procese. Pvz. [36] pristato metodus, kurie aptinka virusus vos tik jiems pasirodžius. Jie naudoja proceso atliekamų sistemos funkcijų kvietimo analizę. Metodų parametrų optimizavimui naudojami GA. Tai leido pasiekti 90% sistemos efektyvumą (tiek taikomųjų programų atpažinta kaip virusai). [37] pasiūlė sprendimą, kuris naudoja genetinį programavimą klasifikavimo taisyklių generavimui, naudojant istorinius duomenis. Metodas naudoja patikimai klasifikuotus duomenis kaip tinkamumo funkciją ir gali tiksliai klasifikuoti tinklinius įsilaužimus[28]. [27] siūlo GA paremtą kenkimo programinės įrangos evoliucijos prognozavimo modelį. Modelyje GA imituoja evoliucijos procesus kenkimo

programinės įrangos vystymosi tendencijoms nustatyti. Modelio tinkamumas vertinamas pasitelkiant istorinius duomenis (naudojant jau žinomą kenkimo programinę įrangą).

4. Windows API

Microsoft Windows taikomųjų programų programavimo sąsaja (API) skirta panaudoti visoms įmanomoms Windows šeimos operacinių sistemų galimybėms. Naudojant API galima kurti programas, kurios veikia visose Windows operacinėse sistemose, atsižvelgiant į kiekvienos teikiamas galimybes.

API versijos:

- **Win16** yra pati pirmoji, pasirodžiusi kartu si 16 bitu Windows operacine sistema. Jos funkcijos įgyvendintos šiuose sistemos branduoliuose: *kernel.exe, user.exe, gdi.exe*.
- Toliau sekė **Win32**. Ją visiškai panaudojo Windows NT sistemoje. Branduolį sudarė kernel32.dll, user32.dll ir gdi32.dll.
- Paskutinė versija – **Win64.** Skirta palaikyti 64 bitu sistemas.

Visa API susideda iš kelių pagrindinių komponentų, kurie atsakingi už tam tikrų sistemos funkcijų atlikimą. Nagrinėjama Win32 API.

Pagrindinės paslaugos: leidžia prieiti prie pagrindinių turimų sistemos resursų. Tai yra rinkmenų sistema, įrenginiai, procesai, klaidų apdorojimas. Šios funkcijos įdėtos į kernel32.dll ir advapi32.dll.

Tinklo paslaugos: leidžia naudotis įvairiausiomis sistemos galimybėmis kuriant ir valdant tinklą.

Grafinių įtaisų sąsaja: leidžia naudotis funkcijomis, kurios išveda informaciją į monitorius, spausdintuvus ir kitus išvesties įrenginius. Įgyvendinta gdi32.dll.

Vartotojo sąsaja: leidžia kurti ir valdyti ekrano langus bei paprastus valdiklius (tokius kaip mygtukai arba slinkties juostos), naudotis pelės bei klaviatūros funkcionalumu. Funkcionalumas įdėtas į user32.dll. Windows XP operacinėje sistemoje pagrindiniai valdikliai perkelti i comctl32.dll.

4.1. Windows API perėmimas

Windows API funkcijų perėmimas reikalauja labai gilaus operacinės sistemos supratimo. Kadangi tai susiję su sistemoje veikiančių procesų atminties modifikavimu, reikia būti labai tiksliam. Žinoma, visa tai galioja jeigu programuotojas funkcijų perėmimo biblioteką rašo nuo nulio. Kai naudojamos trečiųjų šalių siūlomos – rizika, jog sistema bus apgadinta arba veiks nekorektiškai – smarkiai sumažėja[8].

Bibliotekų pasirinkimą nulemia jų panaudojimo reikalavimai. Šiuo atveju pasirinktas metodas bus naudojamas realaus laiko failų nuskaitymui (angl. On-Access Scan)[8].

Metodas turi turėti galimybę perimti funkcijas kurias operacinė sistema naudoja darbui su failais. Windows operacinėje sistemoje tai atlieka *kernel32.dll* bibliotekos funkcijos *CreateFile*, *OpenFile*, *ReadFile* ir kitos[8].

Antras reikalavimas – kai funkcijos jau perimtos, sistemos veikimo spartos pakitimai turi būti minimalūs. Tai galioja ir pačios funkcijos iškvietai ir pačio metodo įdiegimui į norimą procesą[8].

Trečias reikalavimas – metodai (biblioteka) turi būti platinami pagal atvirojo kodo licenciją (kadangi failų nuskaitymas bus naudojant ClamAV duomenų bazę (ji platinama pagal AK (angl. *GPL*) licenciją) [8].

4.2. SetWindowsHookEx funkcija

Šį metodą suteikia Windows API funkcionalumas. Pagal aprašymą jis skirtas funkcijai, esančiai mūsų bibliotekoje į sistemos funkcijų grandinę įterpti. Todėl galima stebėti tam tikrus sistemos įvykius. Vienintelis būtinas reikalavimas naudojant šį mechanizmą: funkcijos pabaigoje iškviesti *CallNextHookEx*. Taip užtikrinsime, kad funkcijos kvietimas nepasibaigs pas mus, jį gaus ir kitos programos, kurios naudoja *SetWindowsHookEx*. Galime stebėti vieno pasirinkto arba visus šiuo metu sistemoje vykdomus procesus. Galimi stebėti įvykiai:

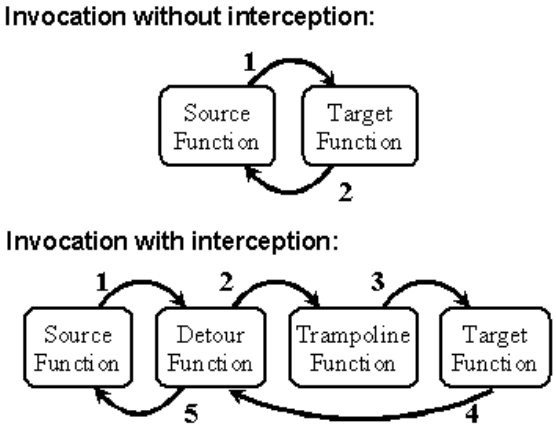
- Klaviatūros veiksmai.
- Pelės veiksmai.
- Dialogų veiksmai.
- Žinutės (angl. *Messages*), siunčiamos ekrano langams.
- Žinutės po jų apdorojimo.

SetWindowsHookEx metodo funkcionalumo pakanka tik stebėti sistemoje naudojamas žinutes. Naudojant jas neįmanoma perimti informacijos, susijusios su darbu failų sistemoje[8].

4.3. Microsoft Detours biblioteka

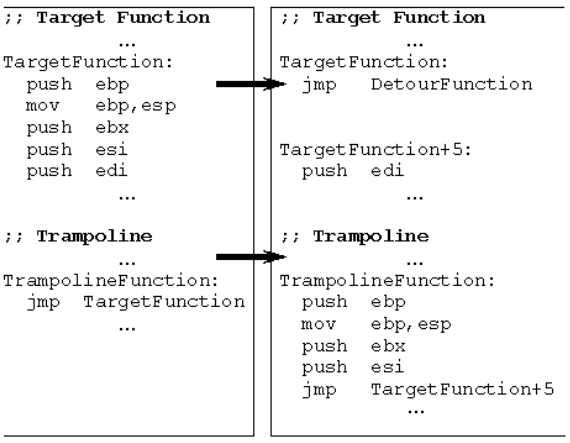
Detours yra biblioteka skirta Win32 funkcijoms perimti. Autoriai – Microsoft tyrimų laboratorijos darbuotojai. Veikimo principas: ji pakeičia pačius pirmuosius funkcijos vykdymo baitus į vartotojo sukurtą funkciją, kuri pakeičia originalą. Originalios funkcijos kvietimas apsaugotas su nukreipiančiąja funkcija (angl. *Trampoline*). Iškvietus perimtą funkciją, kvietimas perduodamas į perrašytą jos variantą. Kai atliekamos papildomos operacijos, dėl kurių buvo perimta

funkcija, kviečiama originali funkcija. Tuomet ir panaudojamas nukreipiančiosios funkcijos išsaugotos originalios funkcijos kvietimas (tai atliekama dėl to, kad vėl neiškviestume perrašytos funkcijos ir nepapultume į amžiną ciklą). 5 pav. parodoma kaip vyksta perrašytos funkcijos kvietimas ir sąsaja su originaliąja prieš ir po perėmimo[8].



5 pav. Kreipimasis prieš ir po perėmimo[8]

Detours biblioteka funkcijas perima perrašydama procesuose esančias jų kvietimo lenteles. Kiekvienai funkcijai biblioteka iš tikrųjų perrašo dvi funkcijas. Viena kviečiama vietoj tikrosios, o kita – aktyvuojama nukreipiančiosios[8].



6 pav. Batuto ir originalios funkcijos programinis kodas prieš ir po Detours bibliotekos įterpimo (kairė ir dešinė)[8]

6 pav. pateiktas pavyzdys parodo originalios ir perimtos funkcijos kodą atmintyje. Visų pirma Detours išskiria atminties nukreipiančiajai funkcijai. Tuomet į jos vietą nukopijuojami mažiausiai 5 baitai originalios funkcijos (tai minimalus kiekis baitų, kai galima atlikti besąlyginę *jmp* komandą). Jei funkcija-taikinys yra trumpesnė nei 5 baitai – Detours biblioteka grąžina klaidą[8].

Detours biblioteka leidžia perimti bet kokios bibliotekos ir bet kokią funkciją. Versija 32 bitų sistemoms platinama nemokamai (skirta nekomerciniam naudojimui), o 64 bitų – mokama (vienkartinis licencijos mokestis – 10 $)[8].

4.4. EasyHook biblioteka

Šis projektas palaiko nekontroliuojamo (angl. *Unmanaged*) programinio kodo perėmimą naudojant kontroliuojamą (angl. *Managed*) kodą. Funkcionalumas pasiekiamas naudojant C# aplinką bei naujas operacines sistemas (Windows 2000 SP4, Windows XP x64, Windows Vista x64 ir Windows Server 2008 x64) [8].

EasyHook biblioteka pateikia keletą naujų dalykų. Visų pirma ji užtikrina, kad perimtoje programoje neliktų jokių resursų ir atminties šiukšlių. Biblioteka taip pat leidžia naudoti kontroliuojamas dorokles (angl. *Handler*) skirtas nekontroliuojamoms API funkcijoms perimti. Tai leidžia naudotis daugybe kontroliuojamo kodo suteikiamų galimybių: .NET tolimų procesų bendravimo funkcionalumu (angl. *.NET Remoting*), WCF (funkcionalumas, skirtas kurti, sujungtas, orientuotas į paslaugas programas) ir WPF (vartotojo sąsajos atvaizdavimui). Taip pat svarbu paminėti, kad EasyHook leidžia su ta pačia programa (angl. *Assemblie*) perimti 32 ir 64 bitų procesuose esančias funkcijas naudojant 32 ir 64 bitų procesus[8].

EasyHook biblioteka stabilią versiją pasiekė 2009m. kovo 8d. Bet kol kas turi keletą smulkių trūkumų susijusiu su kontroliuojamu kodu, kurie bus ištaisyti kitoje versijoje[8].

4.5. Bibliotekų testavimas

Bibliotekos testuojamos norint išsiaiškinti, kuri biblioteka turi mažiausią poveikį perimtų procesų veikimo spartai. Tai yra labai svarbu, kadangi pasirinkta biblioteka bus naudojama realaus laiko failų nuskaitymo sistemoje. Todėl poveikis turi būti minimalus, tiek kuriant naujus procesus, tiek kviečiant perimtas funkcijas[8].

Norint nustatyti perimtos programos pradžios pokyčius, visų pirma išmatuojamas jos įprastinės pradžios laikas. Tam pasirinktas Git (versijų kontrolės sistema) teikiamas funkcionalumas – komandinės eilutės komanda *time*. Ji įvykdo pateiktą komandų grandinę (angl. *Pipeline*) ir atspausdina ekrane sugaištą realų bei procesoriaus laiką[8].

Kiekvienai iš bibliotekų parašytos programos-serveriai. Jie palaiko komandinės eilutės formatą. Kviečiant abi programas nurodoma ką reikia atlikti ir su kokia taikomąja programa. Jei duodama komanda *run* – nurodyta taikomoji programa yra tik paleidžiama ir laukiama, kol naujas procesas baigs savo darbą. Jei *hook* – į nurodytą taikomąją programą įterpiama viena iš bibliotekų (priklausomai nuo serverio) ir perima funkcijos *CreateFile* kvietimą. 7 pav. pateiktas taikomosios

programos, kuri naudojama bibliotekų įsiskverbimo spartai nustatyti, programinis kodas. Kaip matome, programa neatlieka jokių užduočių, tiesiog pradėjusi veikti iškart baigia darbą. Tai užtikrina, kad jos veikimo trukmės nelems vykdomų komandų trukmė[8].

```
#include <tchar.h>

int _tmain(int argc, _TCHAR* argv[])
{
    return 0;
}
```

7 pav. Taikomosios programos, kuri naudojama bibliotekų įsiskverbimo greičiui nustatyti, programinis kodas[8]

8 pav. pateiktos komandos, kurios naudotos nustatyti programos pradžios, veikimo ir užsidarymo spartai naudojant Detours biblioteką. Komandos kartojamos 10 kartų, kad būtų gauti kuo įvairesni rezultatai. Identiškos komandos naudojamos EasyHook testavimui, tik vietoje *DetoursHookCenter.exe* naudojama *FileMon.exe* (EasyHook serveris) programa[8].

```
clear
echo Detours bibliotekos paleidimo testas.
echo Testas bus kartojamas 10 kartu.

for ((  i = 1 ;  i <= 10;  i++  ))
do
    echo
    echo
    echo Bandymas Nr.: $i
    echo RUN
    time DetoursHookCenter.exe run HookDllLoadingTest
    echo
    echo HOOK
    time DetoursHookCenter.exe hook HookDllLoadingTest
done
```

8 pav. Detours bibliotekos įsiskverbimo testavimo komandos[8]

Testuojant perimtų funkcijų veikimo spartos pokyčius naudojama kita taikomoji programa. Ji taip pat 10 kartų atlieka failo kūrimo testą. Pats testas: kuriamas objektas *CFile* su nuoroda, kad failas būtų sukurtas iš naujo. Tokiu būdu iškviečiama *CreateFile* funkcija. Visa tai atliekama 20000 kartų, laikas sumuojamas ir dalinamas iš 20000. Taip gaunamas laikas sugaištas *CFile* objektui sukurti (kartu ir *CreateFile* funkcijai iškviesti). 9 pav. pateiktas šios taikomosios programos programinis kodas[8].

```
for(int k = 0; k < 10; k++)
{
    time_t start, stop;
    int nCount = 20000;

    time(&start);

    for(int i = 0; i < nCount; ++i)
    {
        CFile file(_T("Test.txt"), CFile::modeCreate);
    }

    time(&stop);

    double dDiff = difftime(stop, start);
    printf("Bendras laikas     %.5f s. \n", dDiff);
    double dOneFile = dDiff / nCount;
    printf("Vieno failo laikas %.5f s. \n\n", dOneFile);
}
```

9 pav. CreateFile kvietimo testas[8]

Funkcijos kvietimo testui taip pat naudoti bibliotekų serveriai su komandomis *run* ir *hook*. 10 pav. pateiktos komandos, naudotos EasyHook bibliotekos testavimui.

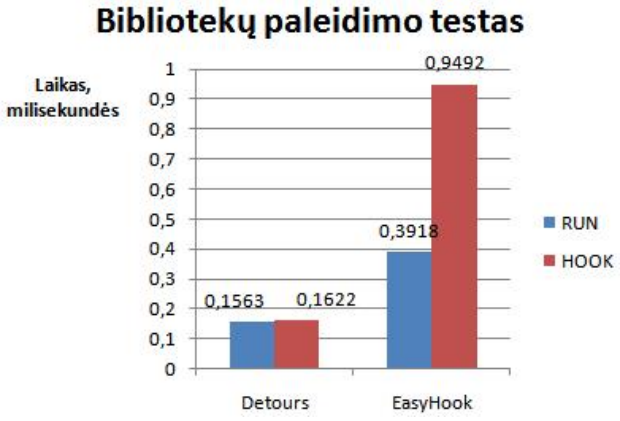
```
clear
echo EasyHook bibliotekos veikimo testas.
echo RUN
FileMon.exe run FileUsage.exe
echo
echo HOOK
FileMon.exe hook FileUsage.exe
```

10 pav. EasyHook bibliotekos CreateFile kvietimo testo komandos[8]

4.6. Testavimo rezultatai

Testai atlikti Windows XP sistemoje. Kompiuterio duomenys: Intel(R) T2250 procesorius, 3 GB operatyvioji atmintis.

11 pav. pateiktas grafikas, vaizduojantis bibliotekų įtaką pradedančiai veikti programai.

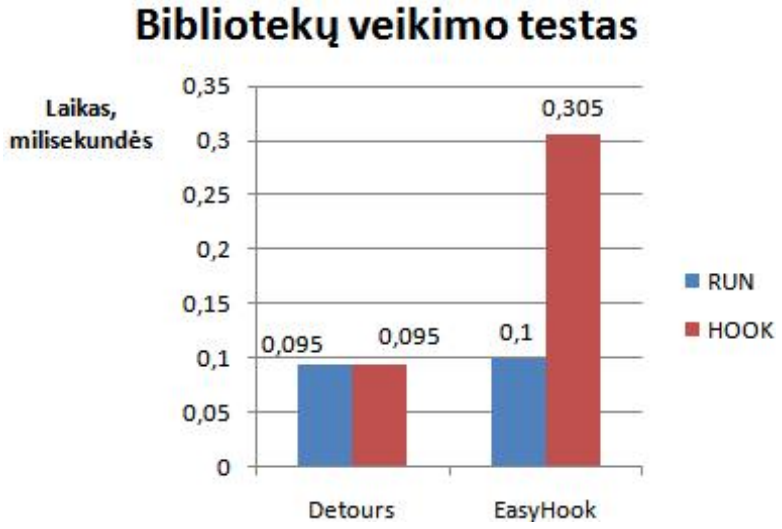


11 pav. Bibliotekų paleidimo testas[8]

Naudojant Detours biblioteką taikomosios programos pradžios trukmė pasikeičia labai nežymiai. Nuo 0,1563 iki 0,1622 milisekundės. Tai įtakoja tik 3,77 % sulėtėjimą. Naudojant

EasyHook biblioteką rezultatai skiriasi net keletą kartų. Pradedant be bibliotekos – 0,3918 milisekundės, o su EasyHook – 0,9492. Tai sudaro 142,27 % sulėtėjimą. Tai reiškia, kad EasyHook biblioteka, programos pradžios trukmę prailgino šiek tiek daugiau nei dvigubai[8].

12 pav. pateiktas funkcijų veikimas įprastu režimu ir perėmus jas su testuojamomis bibliotekomis.



12 pav. Bibliotekų veikimo testas[8]

Kaip matome, Detours biblioteka neturėjo jokios įtakos funkcijos vykdymo trukmei, o EasyHook biblioteka pateikė kitokius rezultatus. Funkcijos vykdymo trukmė nuo 0,1 milisekundės pailgėjo iki 0,305 milisekundės. Tai sudaro 305 % pradinio laiko arba 3 kartų sulėtėjimą[8].

Tokių testų rezultatų buvo galima tikėtis, nes EasyHook bibliotekos dokumentacijoje neaptariama jos sparta. Tuo tarpu Detours dokumentacijoje bibliotekos sparta yra pabrėžiama. 13 pav. pateiktas Detours bibliotekos spartos palyginimas su kitais API perėmimo mechanizmais[8].

Interception Technique	**Intercepted Function**	
	Empty Function	`CoCreate-Instance`
Direct	0.113µs	14.836µs
Call Replacement	0.143µs	15.193µs
DLL Redirection	0.143µs	15.193µs
Detours Library	0.145µs	15.194µs
Breakpoint Trap	229.564µs	265.851µs

13 pav. Įsiskverbimo metodų palyginimas[8]

Šiame teste buvo palygintas tuščios funkcijos ir *CoCreateInstance* veikimas naudojant įvairias įsiskverbimo technikas. Testas atliktas kompiuteryje su 200 MHz Pentium Pro procesoriumi (Hunt *et al.* 1999). Kaip matyti, Detours naudojimas labai mažai trukmės atžvilgiu skiriasi nuo metodų, kurie pripažinti kaip greičiausi[8].

4.7. Tyrimai Windows API perėmimo srityje

Beieškant informacijos apie minėtas bibliotekas paaiškėjo, jog tyrimai šioje srityje nėra labai plėtojami. Windows API perėmimo metodų ir bibliotekų palyginimų yra labai mažai. Daugiausia - pačių gamintojų pristatomieji straipsniai, apžvelgiantys gerąsias savybes[8].

4.8. Windows API perėmimo apžvalgos išvados

1. Atlikus perėmimo metodų apžvalgą, nustatyta, kad perimti darbui su failų sistema skirtas funkcijas gali tik Detours ir EasyHook bibliotekos. Windows API pateikiama SetWindowsHookEx funkcija tinkama tik sistemoje perduodamoms žinutėms ir jų grandinėms perimti[8].

2. Atlikus EasyHook ir Detours bibliotekų testavimą paaiškėjo, kad EasyHook labai stipriai sulėtina tiek programos pradžią, tiek perimtų funkcijų vykdymo trukmę. Tuo tarpu Detours biblioteka turi labai nežymią arba išvis jokios įtakos minėtiems procesams. Šie rezultatai lėmė Detours bibliotekos pasirinkimą tolesnėje praktinėje dalyje[8].

5. Realiojo laiko nuskaitymo taikomoji aplikacija

Aplikacija, kuriama praktinėje dalyje, turi turėti daugumą antivirusinių programų turimų funkcijų:

- Realiojo laiko rinkmenų sistemos stebėjimas.
- Informacija apie dabartinę programos būseną.
- Informacija apie virusų duomenų bazę.
- Paprastas duomenų bazių atnaujinimas.
- Galimybė nuskaityti operatyviąją atmintį
- Galimybė nuskaityti pasirinktą rinkmeną arba katalogą.
- Galimybė ištrinti arba perkelti užkrėstą rinkmeną į karantino katalogą.
- Lengvas programos nuostatų keitimas.
- Duomenų bazių atnaujinimo bei pasirinktų rinkmenų arba katalogų nuskaitymo įdėjimas į užduočių planuoklį.

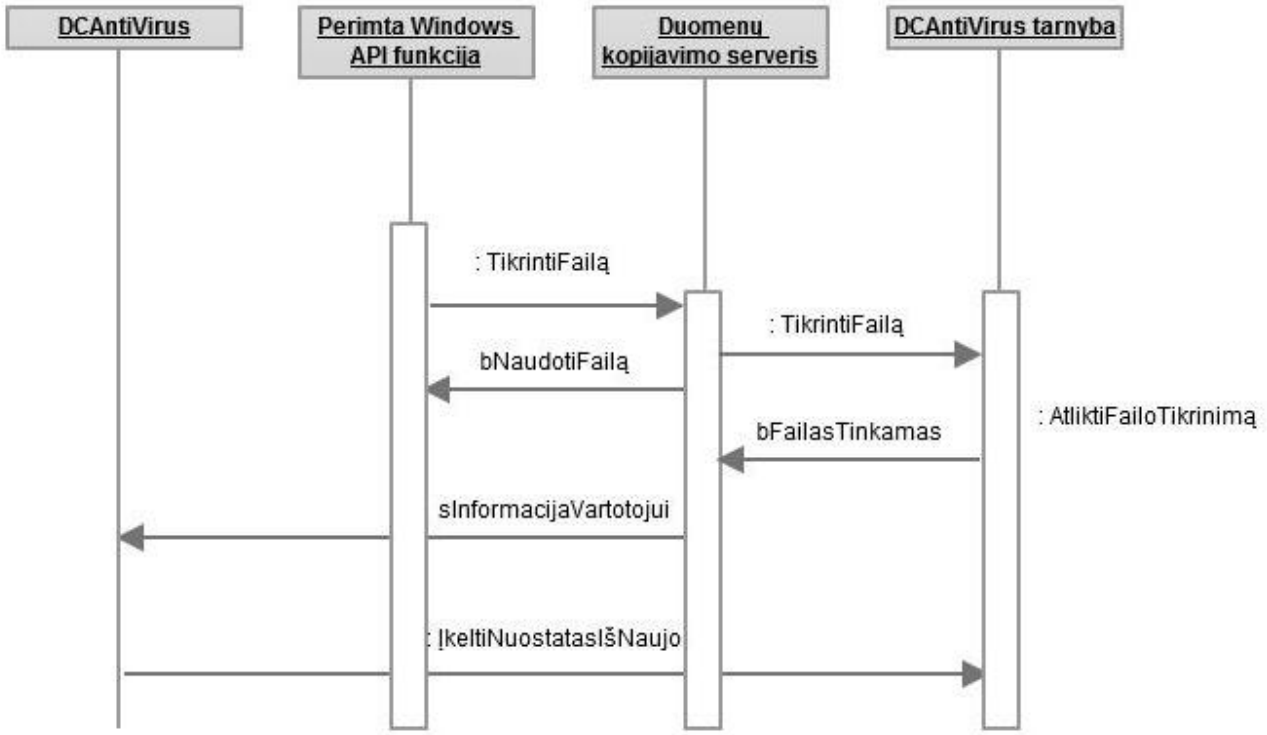
Programa rašoma naudojant C++ programavimo kalbą. Palengvinant užduotį naudojamos trečiųjų šalių pateikiamos bibliotekos.

- Windows API funkcijų perėmimo biblioteka Detours (jau aptarta ankstesniame skyriuje).
- Standartinių C++ funkcijų biblioteka STLport[38].

- Kriptografijos funkcijų biblioteka OpenSSL[29].
- ClamAV antivirusinės programos failo skenavimo biblioteka[7].

Kituose skyriuose bus aptartos kiekviena iš minėtų bibliotekų (naudojamas funkcionalumas bei licencijos).

14 paveiksle pateikiama nuskaitymo sistemos sekų diagrama. Joje parodoma kokia tvarka rinkmena siunčiama patikrinimui bei kokia informaciją gauna kiekvienas sistemos modulis.



14 pav. Sistemos sekų diagrama

5.1. Realiojo laiko rinkmenų sistemos stebėjimas

5.1.1. Rinkmenų palietimo stebėjimas

Norint įgyvendinti šį reikalavimą reikia žinoti kada failų išdėstymo sistemoje esanti rinkmena yra paliečiama (atidaroma, skaitoma, rašoma, trinama, kuriama). Darbui su rinkmenomis Windows API pateikia funkciją *CreateFile(...)*[31]. Ši funkcija sukuria, atidaro arba trina rinkmeną, direktoriją, fizinį diską, komandų grandinę ar pultą. Grąžinamas valdymo objektas, kuris yra skirtas darbui su pasirinkta rinkmena. Funkcijos aprašas:

HANDLE CreateFile(LPCTSTR lpFileName,
DWORD dwDesiredAccess,
DWORD dwShareMode,
LPSECURITY_ATTRIBUTES lpSecurityAttributes,

DWORD dwCreationDispostion ,

DWORD dwFlagsAndAttributes,

HANDLE hTemplateFile);

lpFileName – perduodamas parametras, kuris nusako reikalaujamo objekto (failo, disko, direktorijos) vardą.

deDesiredAcces – perduodamas parametras, kuris nusako kokio priėjimo reikės prie objekto. Tai gali būti skaitymo, rašymo arba tiesiog užklausos teisės.

dwShareMode – perduodamas parametras, kuris nusako kokios teises bus paliktos kitiems procesams, jei jie kreipsis į tą patį objektą.

lpSecurityAttributes – perduodama struktūra, kuri nusako objekto saugumo lygį.

dwCreationDispostion – perduodamas parametras, kuris nusako, kokie veiksmai turi būti atlikti kai failas jau egzistuoja arba jo nėra.

dwFlagsAndAttributes – perduodamas parametras, kuris nusako, su kokiais atributais bus sukurta rinkmena.

hTemplateFile – perduodamas failo šablonas.

Jei funkcijos kvietimas pavyksta, ji grąžina valdymo objektą, jei ne – grąžina INVALID_HANDLE_VALUE reikšmę.

Jos realizacija patalpinta kernel32.dll bibliotekoje ir turi dvi versijas: CreateFileA(..) ir CreateFileW(..). Antroji skirta programoms, kurios palaiko UNICODE, o pirmoji – kurios nepalaiko.

Funkcijos perėmimui naudota biblioteka Detours. Jos aprašymas, bei privalumai pateikti ankstesniame skyriuje. *CreateFile(...)* perėmimas aprašytas naujai sukurtoje bibliotekoje *SystemHook.dll*[19]. Joje įgyvendintas perimto failo perdavimas nuskaitymui bei veiksmai, jei failas yra užkrėstas. Perimtos funkcijos *CreateFile(...)* programinis kodas:

```
(1) if(0 != dwDesiredAccess)
   {
       (2) if(NULL == strstr(lpFileName, "\\\\"))
       {
                (3) if(!wnd_utils::Scan(lpFileName))
                {
                      (4) SetLastError(ERROR_ACCESS_DENIED);
                      (5) return INVALID_HANDLE_VALUE;
                 }
       }
    }
(6) return pTrueCreateFile(...);
```

Eilutėje (1) patikrinama ar rinkmena atidaroma skaitymui arba rašymui. Eilutėje (2) patikrinama ar atidaromas objektas yra rinkmena arba direktorija. (3) eilutėje rinkmena siunčiama nuskaitymui. Jei nuskaitymo funkcija pranešė, kad rinkmena užkrėsta, draudžiamas priėjimas prie jos ((4) ir (5) eilutės). Jei rinkmena yra neužkrėsta, įvykdoma (6) eilutė (tikroji *CreateFile(...)* funkcija).

5.1.2. Procesų bendradarbiavimas

Failų nuskaitymo modulis įgyvendintas atskiroje taikomojoje aplikacijoje. Ši aplikacija sistemoje veikia kaip tarnyba (angl. Service). Tokiu būdu užtikrinama, kad nuskaitymo aplikacija startuos dar prieš vartotojui pradedant seansą. Kadangi perimtos funkcijos ir nuskaitymo modulis veikai skirtinguose procesuose, būtina apibrėžti jų tarpusavio bendravimą.

Procesų bendradarbiavimas (angl. Inter-Process Communication (IPC)) nustato kokiais metodais procesai komunikuos tarpusavyje. Aptarsime keletą metodų.

Bendravimas panaudojant kanalus (angl. Pipes). Šis metodas skirtas bendrauti nesusietiems, bei skirtinguose kompiuteriuose esančiuose procesuose. Programa-serveris sukuria kanalą žinomu vardu. Programa-klientas prisijungia prie šio kanalo. Kai įvyksta susijungimas, programos, naudodamos kanalą, gali keistis duomenimis. Duomenų keitimasis vyksta naudojant skaitymo ir rašymo operacijas[17].

Bendravimas panaudojant Windows prievadus (angl. Windows Sockets). Tai nuo protokolo nepriklausomas metodas. Veikia panašiai kaip ir kanalai. Programa-serveris stebi tam tikrą prievadą. Programa-klientas siunčia duomenis į tą patį prievadą[17].

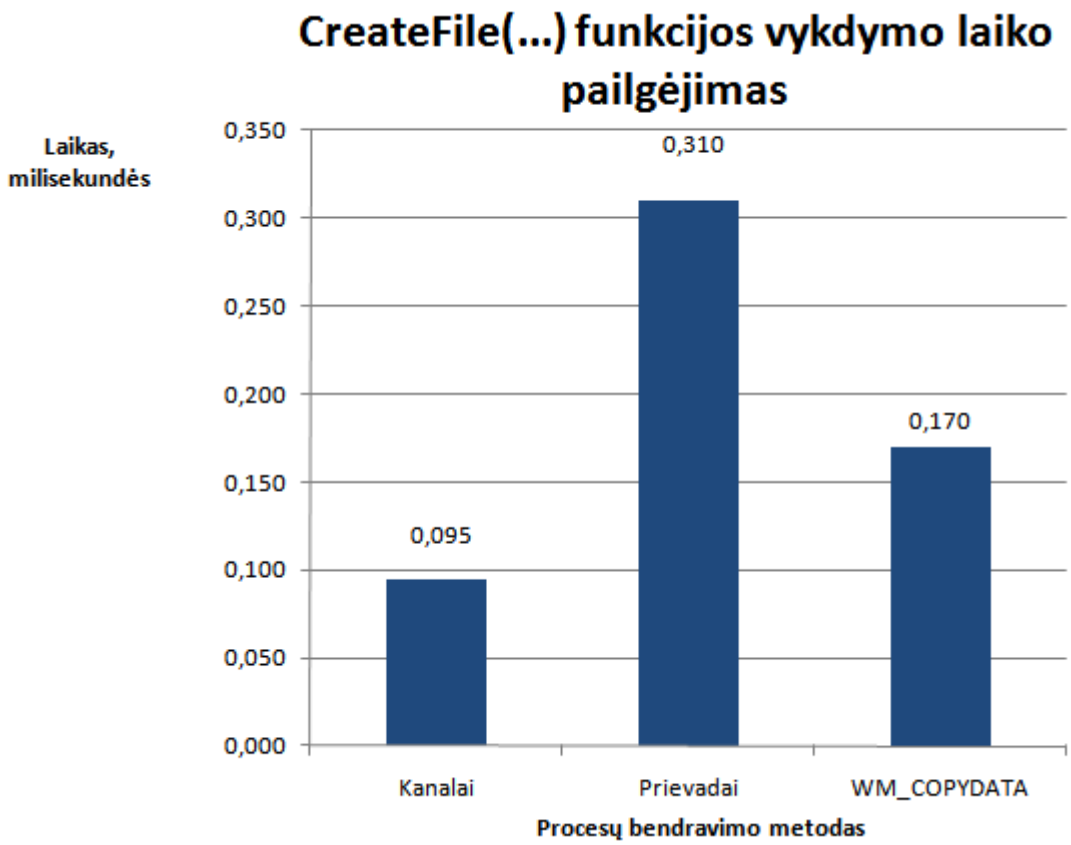
Bendravimas panaudojant duomenų kopijavimą (angl. Data Copy). Šis metodas leidžia iš vienos programos nusiųsti informaciją kitai programai panaudojant WM_COPYDATA žinutę. Programa-serveris turi tiksliai žinoti gaunamos informacijos struktūrą. Programa-klientas privalo nemodifikuoti siunčiamos informacijos kol ją apdoroja programa-serveris. Norint pasiųsti duomenų kopijavimo žinutę reikia žinoti programos-gavėjo pavadinimą[17].

Prieš pasirenkant bendravimo metodą reikia išsiaiškinti kokie reikalavimai jam keliami.

- Metodas privalo užtikrinti spartų komunikavimą tarp procesų.
- Metodas privalo užtikrinti, kad visos užklausos bus apdorotos ir duotas atsakymas ją pasiuntusiam procesui.

Testuojant komunikavimo spartą buvo naudojami tokie patys metodai kaip ir atliekant Windows API bibliotekų spartos testą. Naudota ta pati aplikacija *FileUsage.exe*, kuri 20000 kartų kuria failą, taip iškviesdama *CreateFile(...)* funkciją. Į *FileUsage.exe* buvo įsiskverbta panaudojus

Detours biblioteką. Perimtoje funkcijoje buvo inicijuojamas susijungimas su serveriu, duomenų siuntimas bei atsakymo laukimas. Programa-serveris atsakymą išsiųsdavo vos tik gavęs duomenis. Kiekvienas testas atliktas 10 kartų. Testo rezultatai pateikiami 15 pav.



15 pav. Procesų bendravimo metodų testo rezultatai

Pagal testo rezultatus, funkcijos darbo spartą mažiausiai įtakojo kanalų metodas. Jis lėmė tik 0,095 milisekundės vėlavimą. Tada sekė duomenų kopijavimo metodas – 0,170 milisekundės. Didžiausią įtaką turėjo prievadų metodas – 0,310 milisekundės.

Naudojant kanalų ir prievadų metodus, reikia papildomai organizuoti gautų duomenų apdorojimo eiles. To nereikia naudojant duomenų kopijavimo metodą (duomenų srautai automatiškai valdomi operacinės sistemos).

Taigi atsižvelgiant į testo rezultatus bei duomenų srautų apdorojimo būdus, pasirinkti duomenų kopijavimo ir kanalų metodai. Naudojant duomenų kopijavimo metodą bus užtikrintas korektiškas vienos programos-serverio ir daugybės programų-klientų veikimas, o naudojant kanalus bus kreipiamasi į tarnybą. Tokį sprendimą teko priimti dėl to, kad naujesnėse Windows operacinėse sistemose (Windows Vista bei Windows 7) duomenų kopijavimo metodo įgyvendinimas tarnyboje yra neįmanomas[16]. Microsoft, siekdama padidinti sistemos saugumą, pradėjo griežtai kontroliuoti tarnybų darbą ir jas patalpino į paprastoms taikomosioms aplikacijoms neprieinamą darbalaukį[34].

5.1.3. Failų nuskaitymo tarnyba

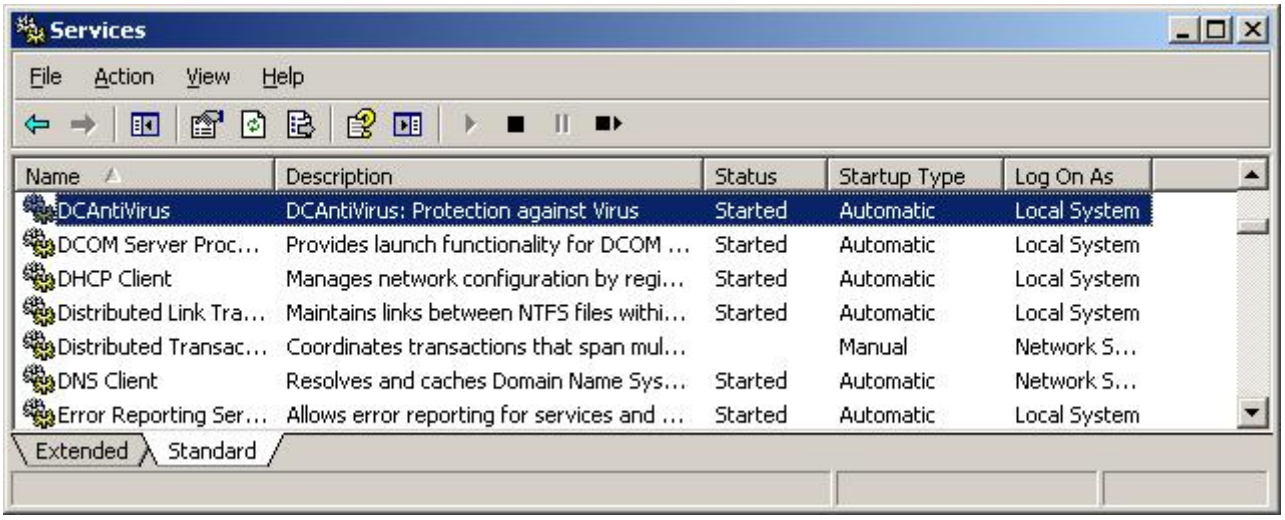
Failų nuskaitymo tarnyba – programa-serveris, kuri savo veikimo metu laukia duomenų kopijavimo būdu siunčiamos informacijos ir ją persiunčia kanalų serveriui[24], gauna atsakymą iš jo ir išsiunčia savo atsakymą. Toliau pateikiamas informacijos apdorojimo pseudokodas.

```
RESULT OnCopyData(WPARAM wParam, LPARAM lParam)
{
    (1) PCOPYDATASTRUCT copy = (PCOPYDATASTRUCT) lParam;
    (2) CSendObj *pData = NULL;
    (3) pData = (CSendObj *)copy->lpData;

    (4) CString sFile = pData->m_sPath;
        CString sVirusName;
    (5) if(!SendFileToService(sFile, sVirusName))
    {
          (6) return VIRUS;
     }
     (7) return OK;
}
```

Eilutėse (1), (2) it (3) iš operatyviosios atminties gaunami siunčiami duomenys. Eilutėje (4) gaunamas failo, kurį reikia tikrinti adresas. (5) eilutėje failas siunčiamas tarnybai ir laukiama jos atsakymo. Jei rinkmena užkrėsta – gražinama VIRUS reikšmė ((6) eilutė), jei rinkmena gera – grąžinama OK reikšmė ((7) eilutė).

Tarnyba pradeda veikti dar prieš vartotojui pradedant sesiją[26]. Pradžioje ji įkelia virusų parašų duomenų bazes. Tuomet paleidžia kanalų serverį, kuris laukia susijungimo. Įvykus susijungimui, atlieka prašomą komandą ir gražina atsakymą. Tarnyba darbą baigia kai išjungiama operacinė sistema. 16 paveikslėlyje pateiktas Windows sistemos tarnybų stebėjimų langas. Tarp jų yra ir DCAntiVirus tarnyba.



16 pav. Windows sistemos tarnybų sąrašas. Pažymėta DCAntiVirus tarnyba

5.1.4. Failų nuskaitymo modulis

Failų nuskaitymo modulis naudojamas rinkmenos patikrinimui. Failas tikrinamas naudojant ClamAV virusų parašų duomenų bazes. Jos yra dvi. Tai pagrindinė (atnaujinama kartą į savaitę), bei kasdieninė (atnaujinama kiekvieną dieną). Siekiant visiško virusų aptikimo tikslumo, naudojama ClamAV siūloma biblioteka, kuri turi rinkmenos nuskaitymo funkciją. Ši biblioteka (*libclamav.dll*) platinama nemokamai. Biblioteka naudoja naujausius metodus ir algoritmus virusų paieškai[18]. Keli iš jų:

- Aho–Corasick eilutės paieškos algoritmas[18].
- Išplėstą Boyer–Moore eilutės paieškos algoritmas[18].

ClamAV bibliotekos palaikomi rinkmenų formatai:

- Vykdomieji failai.
- Elektroninio pašto rinkmenos.
- Populiarūs archyvo formatai.
- Dokumentai (Microsoft Office, PDF, HTML)
- Ir dauguma kitų.

ClamAV bibliotekos nuskaitymo funkcijos aprašas:

*int cl_scanfile(const char *filename,*
*const char **virname,*
*unsigned long int *scanned,*
*const struct cl_engine *engine,*
unsigned int options);

Fukcijai perduodamas rinkmenos adresas (*filename*), rodyklė į kurią įrašomas viruso pavadinimas (*virname*) bei nuskaitymo nuostatos (*options*).

Failų nuskaitymo modulio pseudokodas:

```
bool CScanner::ScanFile(LPCSTR sFile, CString &sVirus, bool bCheckType)
{
        (1) if(!file_utils::FileIsSupported(sFile, m_types))
        {
            return true;
        }

        (2) if(file_utils::FileExistsInInternalDB(sFile))
        {
             return true;
        }

        const char *sVirname;
        (3) if(m_pMainScan->ScanFile(sFile, &sVirname))
        {
             return false;
        }

        (4) if(m_pDailyScan->ScanFile(sFile, &sVirname))
        {
             return false;
        }

        (5) AddFileToInternalDB(sFile);
        return true;
}
```

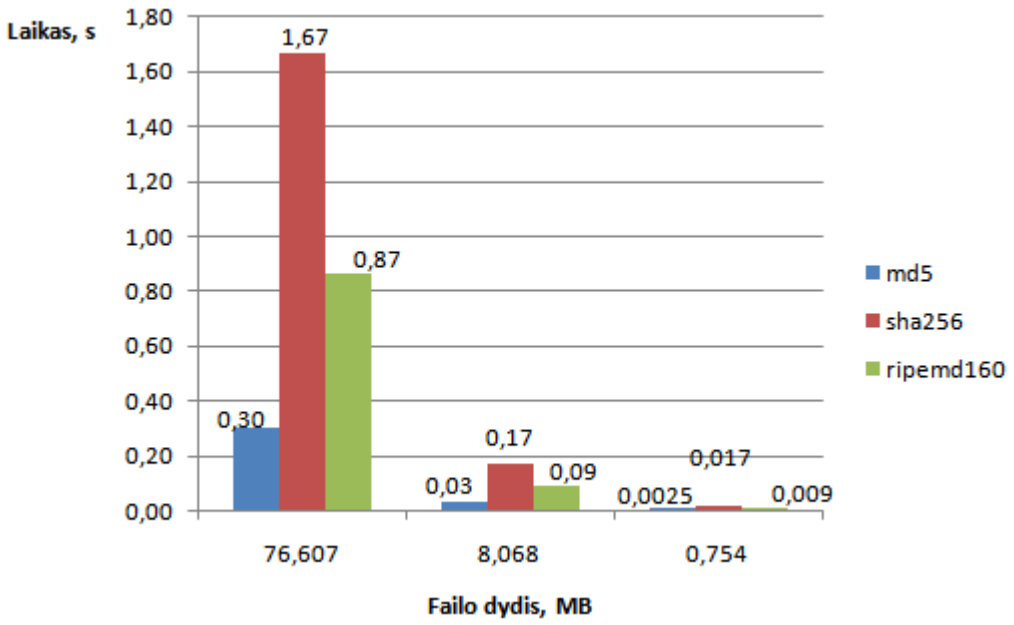
Eilutėje (1) patikrinama ar rinkmenos tipas atitinka vartotojo nustatytas nuskaitymui rinkmenas. Eilutėje (2) patikrinama ar rinkmena nėra išsaugota kaip neužkrėsta. Jei ji egzistuoja vidinėje duomenų bazėje, patikrinama kokie buvo main ir daily duomenų bazių numeriai. Aptikus kurį nors nesutapimą – rinkmena nuskaitoma su nesutapusia virusų parašų duomenų baze. Tokiu būdu užtikrinamas tik reikiamas nuskaitymas. Tuomet eilutėse (3) ir (4) atliekamas rinkmenos nuskaitymas. Jei bent vieno iš jų metu rastas virusas – grąžinama reikšmė, kad failas užkrėstas. Švari rinkmena įrašoma į vidine duomenų bazę ((5) eilutė).

Vidinė duomenų bazė užtikrina, kad vieną kartą patikrintas ir pripažintas švariu failas nebūtų tikrinamas antrą kartą. Pirmiausia reikia suskaičiuoti rinkmenos kontrolinę sumą[5]. Tam atlikti yra daugybė būdų. Tačiau reikia pasirinkti sparčiausią, nes kitų atveju failo atidarymas gali užtrukti ne tik dėl nuskaitymo, bet ir dėl kontrolinės sumos suskaičiavimo[21]. Kontrolinės sumos suskaičiavimui naudojama OpenSSL biblioteka. Ji pateikia daugybę kontrolinės sumos suskaičiavimo metodų. Iš jų pasirinkti trys, su kuriais buvo atliktas spartos testas. Panaudojus OpenSSL biblioteką parašyta taikomoji aplikacija, kuri su kiekvienu metodu suskaičiuodavo trijų

skirtingo dydžio rinkmenų kontrolines sumas. Kiekvienas suskaičiavimas kartotas 10 kartų ir testų rezultatuose pateikiamas jų vidurkis.

1 lentelė. Kontrolinės sumos suskaičiavimo testo rezultatai. Laikas pateiktas sekundėmis

Kontrolinės sumos metodas	Rinkmenos dydis, MB		
	76,607	**8,068**	**0,754**
MD5	0,30	0,03	0,0025
SHA256	1,67	0,17	0,017
RIPEMD160	0,87	0,09	0,009



17 pav. Kontrolinės sumos suskaičiavimo testo grafinė išraiška

Atlikus testą paaiškėjo, kad kontrolinę sumą sparčiausiai suskaičiuodavo MD5 algoritmas. Žinoma, visų algoritmų sparta sumažėdavo dirbant su dideliu failu. Bet visais atvejais MD5 būdavo 3 kartus greitesnis nei RIPEMD160 algoritmas. Tai ir lėmė jo naudojimą rinkmenos nuskaitymo modulyje.

Nuskaitytos rinkmenos, kurios nėra užkrėstos virusu, saugomos STLport bibliotekos pateikiamoje struktūroje *std::Map*. Tai yra susietų duomenų talpykla, užtikrinanti greitą informacijos paiešką, įdėjimą bei paėmimą. Vidinė duomenų bazė saugo failo adreso kontrolinę suma, failo kontrolinę sumą, daily ir main duomenų bazių numerius, failo naudojimo skaitiklį bei failo adresą. Vidinės duomenų bazės įrašo pavyzdys:

44 ec ca 27 f0 88 63 e7 ab 91 91 b1 fc b8 72 60 6d 77 8e 0f 95 44 7e 65 46 55 3e ee a7 09 d0 3c 52 11071 12 C:\WINDOWS\SYSTEM32\CMD.EXE

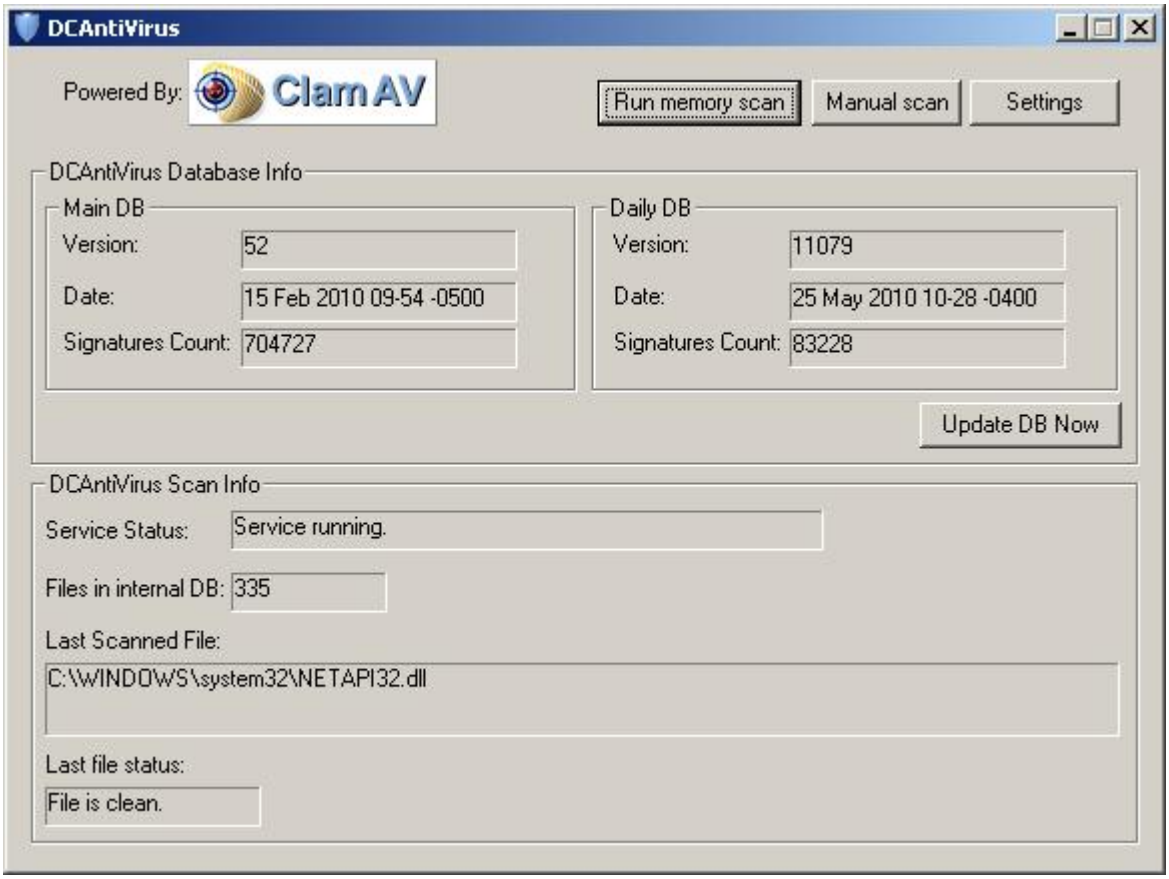
Gerų failų duomenų bazė saugoma *PassData.dat* rinkmenoje. Failų nuskaitymo tarnybos pradžioje šis failas yra užblokuojamas. Taip išvengiama failo sugadinimo arba duomenų pakeitimo. Failo užblokavimas atliekamas panaudojant Windows API funkciją *LockFile(...)*.

5.2. Sistemos dėkle esanti taikomoji aplikacija.

Sistemos dėkle patalpinta programa, kuri skirta vartotojo darbui su tarnyba. Jos pagalba galima keisti tarnybos nuostatas, paleisti katalogų arba rinkmenų nuskaitymą, paliesti atminties nuskaitymą, atnaujinti virusų duomenų bazes, sukurti ir panaikinti planuoklio užduotis.

5.2.1. Pagrindinis programos langas

18 paveikslėlyje pateikiamas pagrindinis programos langas.



18 pav. Pagrindinis programos langas

Pagrindiniame lange pateikiama vartotojui aktuali informacija. Visų pirma pateikiama virusų duomenų bazių informacija (grupėje *DCAntiVirus Database info*). Rodoma pagrindinės ir kasdieninės duomenų bazių numeriai, datos bei turimų parašų kiekis. Grupės apačioje yra mygtukas skirtas duomenų bazių atnaujinimui.

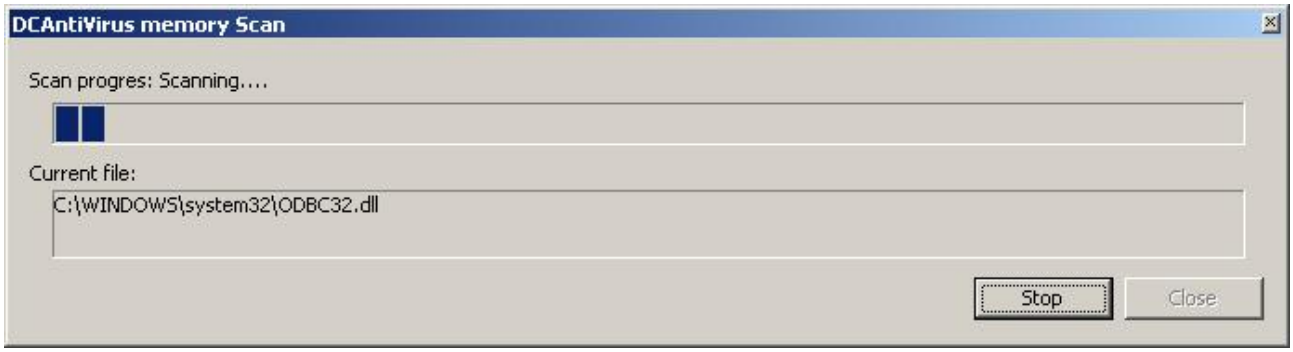
Grupėje *DCAntiVirus Scan info* pateikiama nuskaitymo informacija: tarnybos būsena, vidinėje duomenų bazėje esančių gerų failų kiekis, paskutinis nuskaitytas failas ir jo nuskaitymo rezultatas. Jei failas buvo užkrėstas, failo rezultato laukelyje rodomas viruso pavadinimas.

Viršuje dešinėje yra mygtukai, kurių pagalbą galima atlikti operatyviosios atminties, rinkmenų arba katalogų nuskaitymus, bei keisti programos nuostatas.

19 pav. Pateikiamas programos atvaizdavimas sistemos dėkle.

19 pav. Programa sistemos dėkle

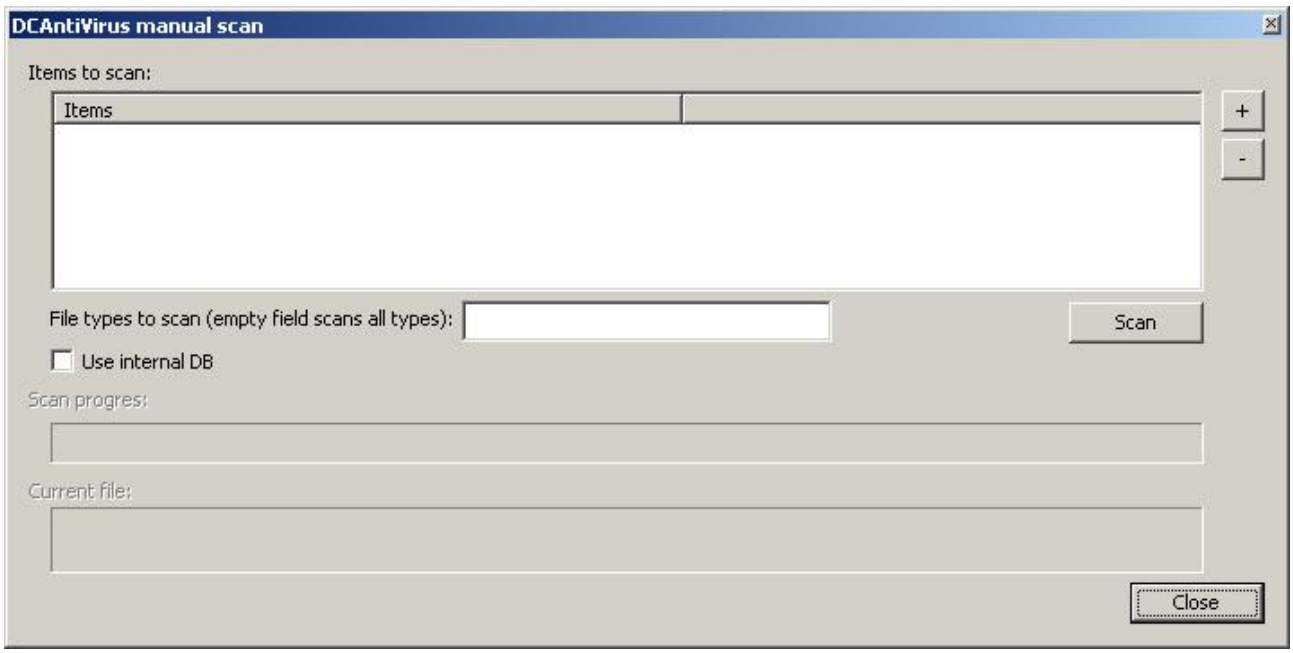
5.2.2. Operatyviosios atminties nuskaitymas



20 pav. Operatyviosios atminties nuskaitymas

20 pav. pateiktas operatyviosios atminties nuskaitymo langas. Jame rodoma nuskaitymo operacijos eiga ir šiuo metu nuskaitomas failas. Yra galimybė sustabdyti nuskaitymą. Baigus arba nutraukus nuskaitymą parodomas rezultatų langas (jis bus aptartas vėliau). Operatyviosios atminties nuskaitymas atliekamas panaudojant ToolHelp metodus[40].

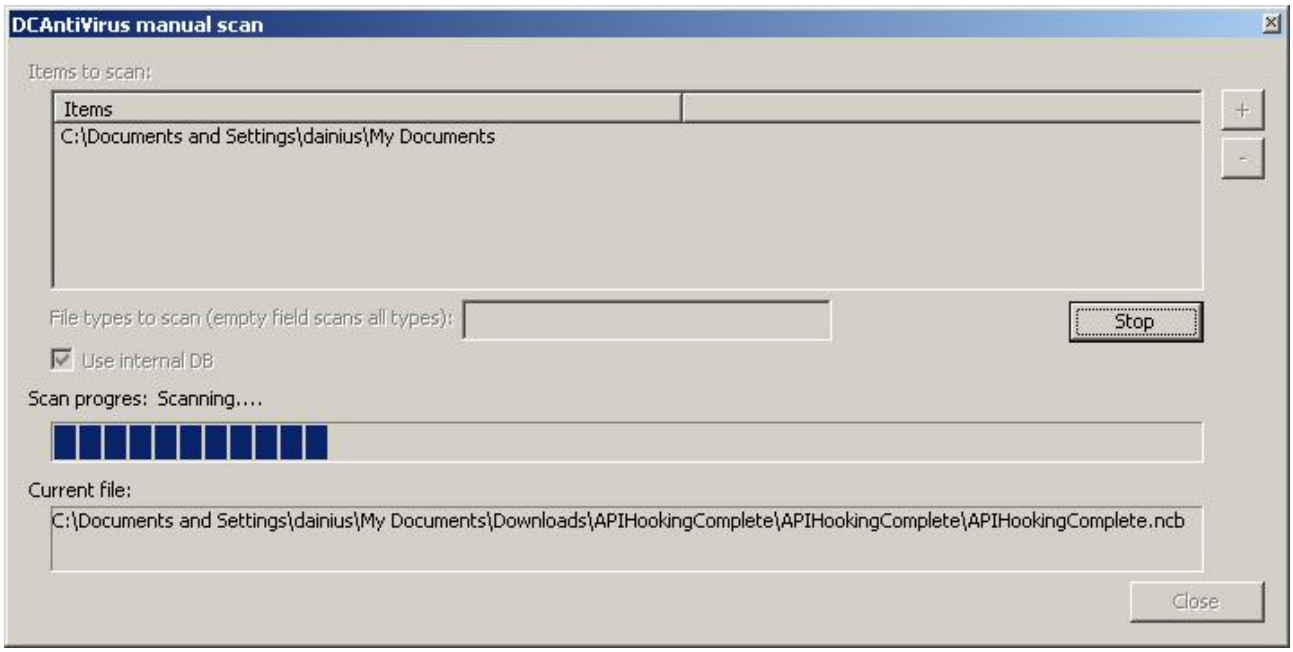
5.2.3. Rinkmenų ir katalogų nuskaitymas pagal pareikalavimą



21 pav. Rinkmenų ir direktorijų nuskaitymo pagal pareikalavimą langas

Šioje programos dalyje galima nurodyti rinkmenas arba katalogus, kuriuos reikia nuskaityti. Tai atliekama su + ir - mygtukais. Galima nurodyti nuskaitomų rinkmenų plėtinius. Pasirinkus *Use internal DB* rinkmenos bus patikrinamos su vidinės duomenų bazės įrašais. Nuskaitymas prasideda paspaudus *Scan*.

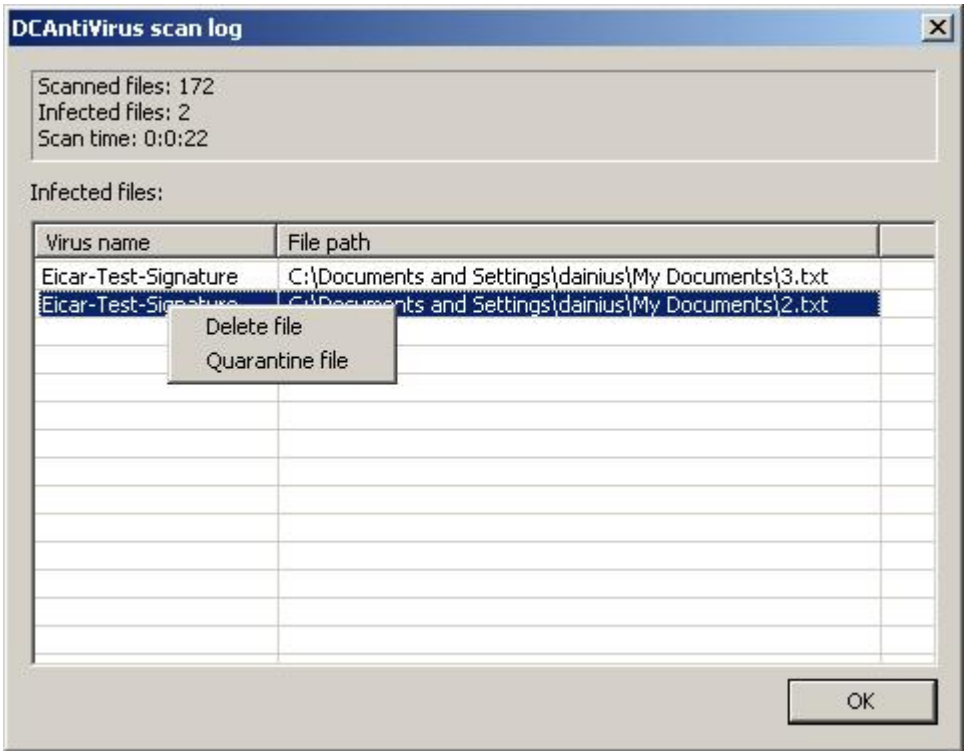
22 paveikslėlyje pateikiamas nuskaitymo pagal pareikalavimą eigos langas.



22 pav. Nuskaitymo pagal pareikalavimą eigos langas

Vykstant pasirinktų rinkmenų arba katalogų nuskaitymui rodoma eiga bei šiuo metu nuskaitomas failas. Paspaudus mygtuką *Stop* arba pasibaigus nuskaitymui parodomas rezultatų langas.

5.2.4. Nuskaitymo rezultatų langas



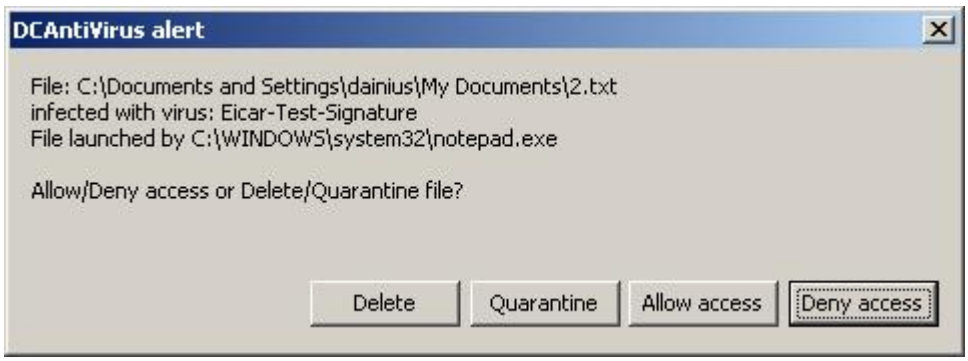
23 pav. Nuskaitymo rezultatų langas

Nuskaitymo rezultatų lange pateikiama vartotojui aktuali informacija: nuskaitytų rinkmenų skaičius, užkrėstų rinkmenų skaičius ir nuskaitymo trukmė. Jei buvo rasta užkrėstų failų pateikiamas jų sąrašas. Paspaudus dešinį pelės klavišą atsiranda kontekstinis meniu (matomas 23 pav.). Jame galima pasirinkti du failo apdorojimo būdus:

- *Delete file*. Rinkmena bus pašalinta iš standžiojo disko.
- *Quarantine file*. Rinkmena bus perkelta į karantino direktoriją.

5.2.5. Realiojo laiko nuskaitymo pranešimas apie užkrėstą rinkmeną

Realiojo nuskaitymo metu stebima visas sistema ir joje paliesti failai. Jei paliestas failas yra užkrėstas, pranešama vartotojui. Jam parodomas 24 pav. pateiktas dialogo langas.

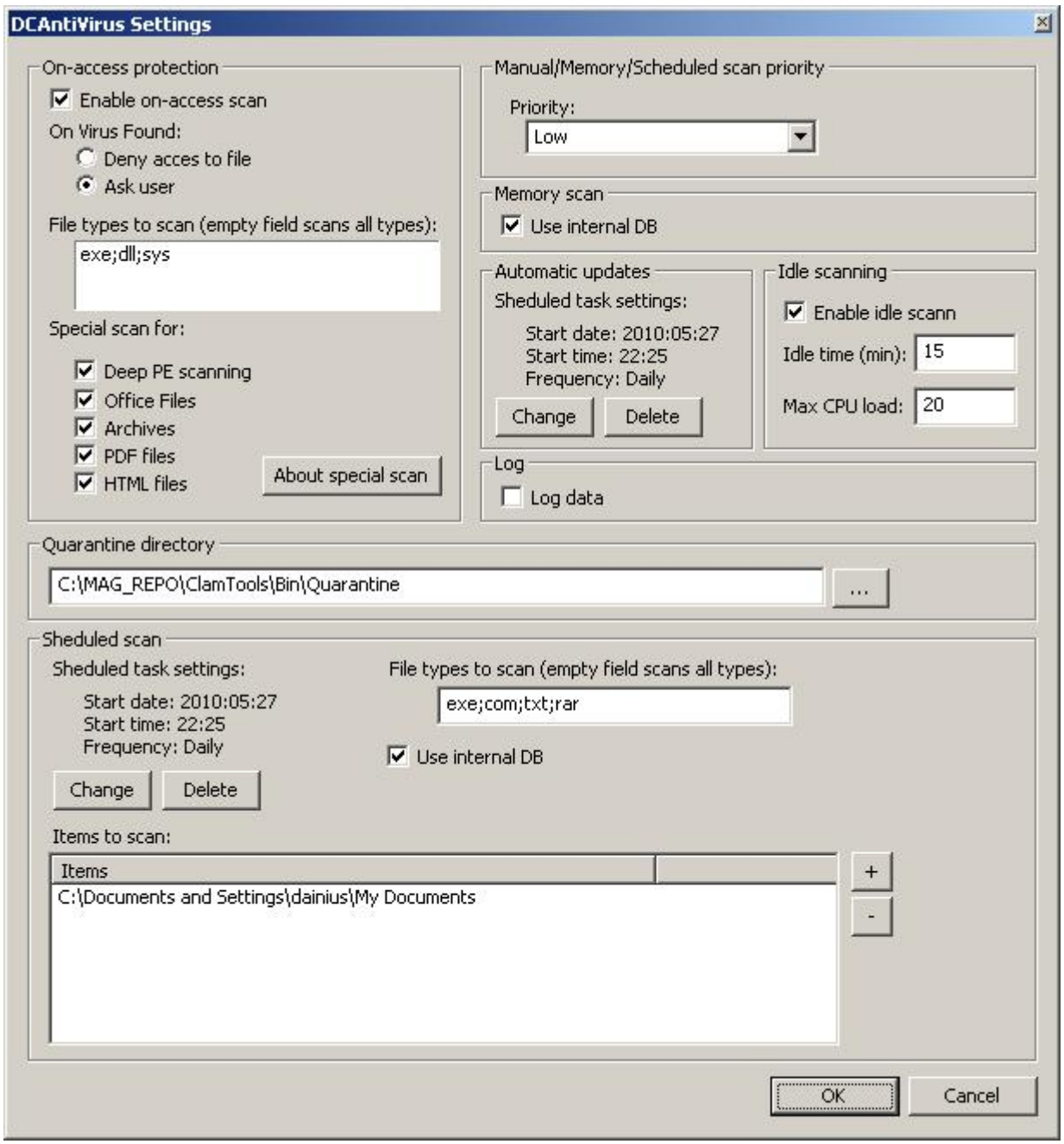


24 pav. Pranešimas apie užkrėstą rinkmeną

Vartotojui pateikiamas užkrėstos rinkmenos adresas, viruso pavadinimas ir rinkmeną norinti atidaryti programa. Vartotojas gali pasirinkti vieną iš keturių veiksmų:

- *Delete.* Rinkmena bus ištrinta iš standžiojo disko.
- *Quarrantine.* Rinkmena bus perkelta i karantino dialogą.
- *Allow access.* Leidžiama programai atidaryti norimą rinkmeną.
- *Deny access.* Draudžiamas priėjimas prie rinkmenos.

5.2.6. Programos nuostatų langas



25 pav. Programos nuostatų langas

Šiame lange galima keisti programos nuostatas. Nuostatos suskirstytos į logines grupes.

On-access protection. Šioje grupėje pateiktos realiojo laiko failų nuskaitymo nuostatos. Galima išjungti arba įjungti nuskaitymas, nustatyti ar programa atmes priėjimą prie rinkmenos ar paklaus vartotojo ką daryti. Vartotojas taip pat gali pasirinkti kurių plėtinių rinkmenos bus nuskaitomos. Atsižvelgiant į *libclamav.dll* bibliotekos teikiamas galimybes, vartotojui leidžiama pasirinkti papildomi failų nuskaitymo algoritmai.

Manual/Memory/Scheduled scan priority. Šioje grupėje galima pasirinkti vartotojo inicijuoto nuskaitymo eigos svarbą. Galimi trys pasirinkimo variantai: *Normal, Low, Lowest.*

Memory scan. Vartotojas gali pasirinkti ar naudoti vidinę duomenų bazę atliekant operatyviosios atminties nuskaitymą.

Automatic updates. Galima keisti automatinio virusų duomenų bazių atnaujinimo nuostatas.

Idle scanning. Galima keisti laisvo nuskaitymo nuostatas.

Log. Galima įjungti arba išjungti papildomos programos darbo informacijos išsaugojimą.

Quarantine directory. Vartotojas gali nurodyti karantino katalogo adresą.

Scheduled scan. Galima keisti planuoto nuskaitymo nuostatas (rinkmenų plėtinių nurodymas, rinkmenų ir katalogų sąrašas)[39].

5.3. Programos testavimas.

Programos testavimas atliekamas palyginant jos nuskaitymo rezultatus su kitomis antivirusinėmis aplikacijomis. Palyginimui naudotos AVG[2] bei Avira[3] nemokamos antivirusinės programos. Visų pirma testuotas programų nuskaitymo pagal pareikalavimą efektyvumas. Visoms programoms buvo nurodytas tas pats katalogas turintis 171 rinkmeną, kurios užima 36 MB standžiojo disko. Sekantis testas skirtas operatyviosios atminties nuskaitymui. 2 lentelėje pateikti testo rezultatai.

2 lentelė. Katalogo nuskaitymo testo rezultatai. Laikas pateikiamas sekundėmis

Programos pavadinimas	DCAntiVirus	AVG	Avira
Katalogo nuskaitymas	26 (12)	12	6
Operatyviosios atminties nuskaitymas	204 (72)	Nėra galimybės	18

Testo rezultatai parodė, kad DCAntiVirus smarkiai atsiliko nuo kitų programų. Lentelėje skliaustuose pateikiamas laikas, kai nuskaitymas atliktas panaudojant vidinę duomenų bazę. Taigi testuose geriausiai pasirodė Avira, o AVG net neturėjo galimybės nuskaityti atmintį.

DCAntiVirus naudojant kaip realiojo laiko nuskaitymo programą ji veikė stabiliai. Tačiau, jei vartotojas naršydavo po katalogus, programa nuskaitydavo visa katalogo turinį. To neatsitikdavo naudojant *AVG* bei *Avira*.

5.4. Trečiųjų šalių bibliotekų apžvalga

Trumpai aptarsime praktinėje dalyje naudojamas trečiųjų šalių bibliotekas. Kaip jau minėjau 4 skyriuje naudojamos šios bibliotekos: Detours, STLPort, OpenSSL ir ClamAV.

Detours. Microsoft gaminys skirtas perimti Windows API funkcijas. 32 bitų versija yra nemokama nekomerciniam naudojimui[8].

STLPort. Standartinių C++ naudojamų funkcijų bei duomenų talpyklų rinkinys. Galima laisvai naudoti net ir komerciniuose gaminiuose[38].

OpenSSL. Standartinių ir populiariausių kriptografijos funkcijų bei algoritmų rinkinys. Gali būti naudojamas net ir komerciniuose gaminiuose[29].

ClamAV. Tai nemokama antivirusinė programa skirta Linux sistemai. Ji taip pat turi versija Windows sistemai. Praktinėje dalyje naudota rinkmenos nuskaitymo biblioteka. Gali būti naudojama tik nekomerciniuose gaminiuose[7].

IŠVADOS

Teorinėje dalyje aptartos virusų aptikimo metodikos, jų raida. Pastebėta, kad naujausios technologijos pritaikomos ir virusų aptikime. Senokai naudojami neuroniniai tinklai. Taip pat pamažu pritaikomi genetiniai algoritmai. Nustatyta, kad virusų aptikimui siūlomas ne tik nuskaitymas pagal virusų parašų duomenų bazę, bet ir panaudojant Windows API funkcijų kvietimo sekas bei Windows API antspaudų patikrinimą.

Kitoje teorinėje dalyje aptartos *Detours* ir *EasyHook* Windows API funkcijų perėmimo bibliotekos. Paaiškėjo, kad sparčiausia iš jų yra Microsoft siūloma *Detours* biblioteka. To ir buvo galima tikėtis. *EasyHook* bibliotekos autoriai niekur neužsimena apie jos spartą, o *Detours* autoriai ją pabrėžia pateiktoje dokumentacijoje. Ši biblioteka buvo pasirinkta antrai praktinei daliai atlikti.

Praktinėje dalyje sukurta taikomoji programa skirta realiojo laiko Windows sistemos apsaugai. Ji stebi sistemoje paliestus failus, juos nuskaito ir praneša vartotojui apie užkrėstas rinkmenas. Programoje naudojamos nemokamos ClamAV virusų parašų duomenų bazės. Siekiant optimizuoti sistemos darbą įgyvendintas gerų failų saugojimas vidinėje duomenų bazėje. Jei failas nuskaitomas ir yra geras, jis išsaugomas vidinėje duomenų bazėje. Jei rinkmena nepakeista – antra kartą ji nebenuskaitoma.

Programa turi visas privalomas antivirusinės programos funkcijas:

- Operatyviosios atminties, rinkmenos ir katalogo nuskaitymas pagal pareikalavimą.
- Galimybė atnaujinti virusų parašų duomenų bazes.
- Galimybė į užduočių planuoklį įdėti pasirinktos rinkmenos arba katalogo nuskaitymą bei virusų parašų duomenų bazių atnaujinimą.

Atlikus testus paaiškėjo, kad programa nepasižymi sparta. Ji smarkiai nusileido nemokamoms AVG bei Avira antivirusinėms programoms. Kadangi Windows API funkcijų perėmimui bei informacijos perdavimui tarp procesų pasirinkti sparčiausi metodai, ilgiausiai užtrunka rinkmenos nuskaitymas.

Tolimesni tyrimai gali būti atliekami apimant konkretesnę virusų aptikimo sritį. Tai gali būti genetinių algoritmų panaudojimas optimizuojant realiojo laiko nuskaitymo sistemas. Taip pat reikia atlikti programos optimizaciją realiojo laiko sistemos rinkmenų stebėjimui.

LITERATŪROS SĄRAŠAS

1. Asaf Shabtai, R. Moskovitch. Detection of malicious code by applying machine learning classifiers on static features: A state-of-the-art survey. Iš: Information security technical report 14, 2009.
2. AVG [interaktyvus] 2010. [žiūrėta 2010 m. Gegužės 29 d.]. Prieiga per internetą: <http://free.avg.com/de-en/homepage>.
3. Avira [interaktyvus] 2010. [žiūrėta 2010 m. Gegužės 29 d.]. Prieiga per internetą: <http://www.free-av.com>.
4. Bert den Boer, A. Bosselaers. Collisions for the Compression Function of MD5. Springer, 1993.
5. Bruce Schneier. Applied Cryptography. John Wiley & Sons, 1996.
6. C. M. Linn, M. Rajagopalan, S. Baker, C. Collberg, S. K. Debray, J. H. Hartman. Protecting Against Unexpected System Calls. Iš: 14th USENIX Security Symposium, 2005.
7. ClamAV [interaktyvus] 2010. [žiūrėta 2010 m. Gegužės 29 d.]. Prieiga per internetą: <http://www.clamav.net/lang/en>.
8. Čeponis Dainius, Radvilavičius L. Windows API funkcijų perėmimo bibliotekų tyrimas, 13-osios Lietuvos jaunųjų mokslininkų konferencijos „Mokslas – Lietuvos ateitis“, įvykusios Vilniuje 2010 m. balandžio mėn. 16 d., pranešimų medžiaga.
9. David Wiliamson. Deconstructing malware: what it is and how to stop it. Iš: Information Security Technical Report. Vol. 9, No. 2, 2004.
10. Ed Skoudis, L. Zeltser. Malware: Fighting Malicious Code. Prentice Hall, 2003.
11. Eitan Menahem, A. Shabtai, L. Rokach. Improving malware detection by applying multi-inducer ensemble. Iš: Computational Statistics and Data Analysis 53, 2009.
12. Essam Al Daoud, Iqbal H. Jebril and Belal Zaqaibeh. Computer Virus Strategies and Detection Methods. Int. J. Open Problems Compt. Math., Vol. 1, No. 2, September 2008.
13. F-Secure Virus Descriptions : Stoned [interaktyvus] 2010. [žiūrėta 2010 m. Gegužės 29 d.]. Prieiga per internetą: <http://www.f-secure.com/v-descs/stoned.shtml>.
14. Gary McGraw. Software security : building security in. Addison Wesley Professional, 2006.
15. Hai Jin, G. Xiang, D. Zou. A guest-transparent file integrity monitoring method in virtualization environment. Iš: Computers and Mathematics with Applications, 2010.
16. Interactive Services [interaktyvus] 2010. [žiūrėta 2010 m. Gegužės 29 d.]. Prieiga per internetą: <http://msdn.microsoft.com/en-us/library/ms683502>.

17. Interprocess Communications [interaktyvus] 2010. [žiūrėta 2010 m. Gegužės 29 d.]. Prieiga per internetą: <http://msdn.microsoft.com/en-us/library/aa365574%28VS.85%29.aspx>.

18. Yevgeniy Miretskiy, A. Das, C. Wright, E. Zadok. Avfs: An On-Access Anti-Virus File System. Iš: 13th USENIX Security Symposium, 2004.

19. John Peloquin. Remote Library Loading [interaktyvus] 2002. [žiūrėta 2010 m. Gegužės 29 d.]. Prieiga per internetą: <http://www.codeguru.com/cpp/w-p/dll/article.php/c3651>.

20. John Viega, Matt Messier. Secure Programing CookBook. O'Reilly, 2003.

21. Jonathan Katz, Y. Lindell. Introduction to Modern Cryptography: Principles and Protocols. Chapman and Hall/CRC, 2007.

22. Marshall D. Abrams, S. Jajodia, H. Podell. Information Security: An Integrated Collection of Essays. Los Alamitos, CA USA, 1995.

23. Meng Zhang, Y.Zhang, L.Hu. A faster algorithm for matching a set of patterns with variable length don't cares. Iš: Information Processing Letters 110, 2010.

24. Multithreaded Pipe Server [interaktyvus] 2010. [žiūrėta 2010 m. Gegužės 29 d.]. Prieiga per internetą: <http://msdn.microsoft.com/en-us/library/aa365588%28v=VS.85%29.aspx>.

25. Named Pipes [interaktyvus] 2010. [žiūrėta 2010 m. Gegužės 29 d.]. Prieiga per internetą: <http://msdn.microsoft.com/en-us/library/aa365590%28VS.85%29.aspx>.

26. Nigel Thompson. Creating a Simple Win32 Service in C++ [interaktyvus] 1995. [žiūrėta 2010 m. Gegužės 29 d.]. Prieiga per internetą: <http://msdn.microsoft.com/en-us/library/ms810429.aspx>.

27. Nikolaj Goranin, A. Čenys, J. Juknius. Extension of the Genetic Algorithm Based Malware Strategy Evolution Forecasting Model for Botnet Strategy Evolution Modeling, in Proc. of NATO RTO Information Systems Technology Panel Symposium, Information Assurance and Cyber Defense (IST-091 / RSY-021), Antalya. Turkey. RTA-NATO, P8-1–P8-20. 2010.

28. Nikolaj Goranin, A. Čenys. Genetinių algoritmų taikymo įsiskverbimų detektavimo sistemose analizė. 12-osios Lietuvos jaunųjų mokslininkų konferencijos „Mokslas – Lietuvos ateitis", įvykusios Vilniuje 2009 m. balandžio mėn. 6 d., pranešimų medžiaga.

29. OpenSSL [interaktyvus] 2010. [žiūrėta 2010 m. Gegužės 29 d.]. Prieiga per internetą: <http://www.openssl.org>.

30. Peter Szor. The art of computer virus research and defense. Hagerstown, February 2005.

31. Prasad Dabak, S. Phadke, M. Borate. Undocumented Windows NT. New York, 1999

32. Roberto Perdisci, A. Lanzi, W. Lee. Classification of packed executables for accurate computer virus detection. Iš: Pattern Recognition Letters 29, 2008.

33. Ross Anderson. Security Engineering - The Book. John Wiley & Sons, Inc., 2001.

34. Security, services and the interactive desktop in Windows [interaktyvus] 2005. [žiūrėta 2010 m. Gegužės 29 d.]. Prieiga per internetą: < http://support.microsoft.com/kb/327618>.

35. Seokwoo Choi, H. Park, H. Lim, T. Han. A static API birthmark for Windows binary executables. Iš: The Journal of Systems and Software 82, 2009.

36. Sherman S.M., Lucas C.K. A generic anti-spyware solution by access control list at kernel level. The Journal of Systems and Software 75, 2005.

37. Syed Bilal Mehdi, A. Tanwani, M. Farooq. IMAD: In-Execution Malware Analysis and Detection. Iš: Genetic and Evolutionary Computation Conference (GECCO), 2009.

38. STLport [interaktyvus] 2008. [žiūrėta 2010 m. Gegužės 29 d.]. Prieiga per internetą: <http://www.stlport.org>.

39. Task Scheduler 1.0 Interfaces. [interaktyvus] 2010. [žiūrėta 2010 m. Gegužės 29 d.]. Prieiga per internetą: <http://msdn.microsoft.com/en-us/library/aa383581%28VS.85%29.aspx>.

40. ToolHelp Functions [interaktyvus] 2010. [žiūrėta 2010 m. Gegužės 29 d.]. Prieiga per internetą: <http://msdn.microsoft.com/en-us/library/aa915058.aspx>.

PRIEDAS

1 PRIEDAS

INFORMATIKA

13-osios Lietuvos jaunųjų mokslininkų konferencijos „Mokslas – Lietuvos ateitis“, įvykusios Vilniuje 2010 m. balandžio mėn. 16 d., medžiaga

WINDOWS API FUNKCIJŲ SEKŲ PERĖMIMO BIBLIOTEKŲ TYRIMAS

Dainius Čeponis [1], **Lukas Radvilavičius** [2]

Vilniaus Gedimino technikos universitetas
El. paštas: [1]*vgtu.dainius@gmail.com;* [2] *lukas@fmf.vgtu.lt*

Santrauka. Nagrinėjamos bibliotekos (angl. Dynamic-link library), skirtos Windows API (angl. Application Programming Interface) funkcijų sekų perėmimui. Sekų perėmimas gali būti panaudojamas įvairiose srityse: siekiant išsiaiškinti operacinės sistemos veikimo principus, norint atlikti programos derinimą (angl. *Debugging*) arba pridėti papildomą funkcionalumą prie jau esamo. Straipsnyje apžvelgiamos Windows API bei trečiųjų šalių pateikiamos priemonės funkcijų sekų perėmimui. Aptariamos jų teikiamos galimybės, panaudojimo specifika. Taip pat atlikti testai siekiant išsiaiškinti kuri biblioteka su užduotimis susitvarko greičiausiai. Tyrimui pasirinkti du nemokami gaminiai: Microsoft Detours ir EasyHook bibliotekos.

Reikšminiai žodžiai: Windows API, funkcijų sekų perėmimas, įsiskverbimo bibliotekos, palyginimas, C++, C#, .NET, Detours, EasyHook, SetWindowsHookEx, Atviras Kodas.

Įvadas

Windows API funkcijų perėmimas reikalauja labai gilaus operacinės sistemos supratimo. Kadangi tai susiję su sistemoje veikiančių procesų atminties modifikavimu, reikia būti labai tiksliam. Žinoma, visa tai galioja jeigu programuotojas funkcijų perėmimo biblioteką rašo nuo nulio. Kai naudojamos trečiųjų šalių siūlomos – rizika, jog sistema bus apgadinta arba veiks nekorektiškai – smarkiai sumažėja.

Bibliotekų pasirinkimą nulemia jų panaudojimo reikalavimai. Šiuo atveju pasirinktas metodas bus naudojamas realaus laiko failų nuskaitymui (angl. On-Access Scan).

Metodas turi turėti galimybę perimti funkcijas, kurias operacinė sistema naudoja darbui su failais. Windows operacinėje sistemoje tai atlieka *kernel32.dll* bibliotekos funkcijos *CreateFile*, *OpenFile*, *ReadFile* ir kitos.

Antras reikalavimas – kai funkcijos jau perimtos, sistemos veikimo spartos pakitimai turi būti minimalūs. Tai galioja ir pačios funkcijos iškvietai ir pačio metodo įdiegimui į norimą procesą.

Trečias reikalavimas – metodai (biblioteka) turi būti platinami pagal atvirojo kodo licenciją (kadangi failai nuskaitomi naudojant ClamAV duomenų bazę (ji platinama pagal AK (angl. *GPL*) licenciją)).

SetWindowsHookEx funkcija

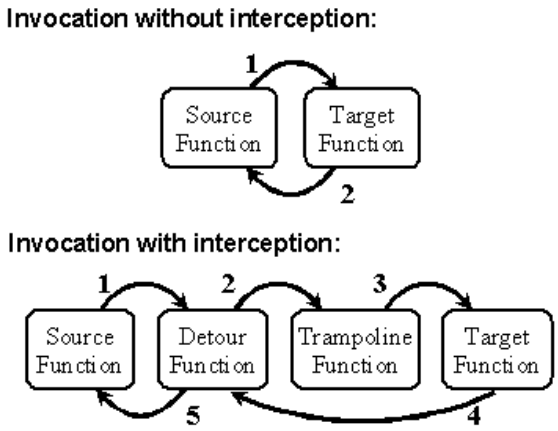
Šį metodą suteikia Windows API funkcionalumas. Pagal aprašymą jis skirtas funkcijai, esančiai mūsų bibliotekoje į sistemos funkcijų grandinę įterpti. Todėl galima stebėti tam tikrus sistemos įvykius. Vienintelis būtinas reikalavimas naudojant šį mechanizmą: funkcijos pabaigoje iškviesti *CallNextHookEx*. Taip užtikrinsime, kad funkcijos kvietimas nepasibaigs pas mus, jį gaus ir kitos programos, kurios naudoja *SetWindowsHookEx* (Čeponis *et al.* 2008). Galime stebėti vieno pasirinkto arba visus šiuo metu sistemoje vykdomus procesus. Galimi stebėti įvykiai:

- Klaviatūros veiksmai.
- Pelės veiksmai.
- Dialogų veiksmai.
- Žinutės (angl. *Messages*), siunčiamos ekrano langams.
- Žinutės po jų apdorojimo.

SetWindowsHookEx metodo funkcionalumo pakanka tik stebėti sistemoje naudojamas žinutes. Naudojant jas neįmanoma perimti informacijos, susijusios su darbu failų sistemoje.

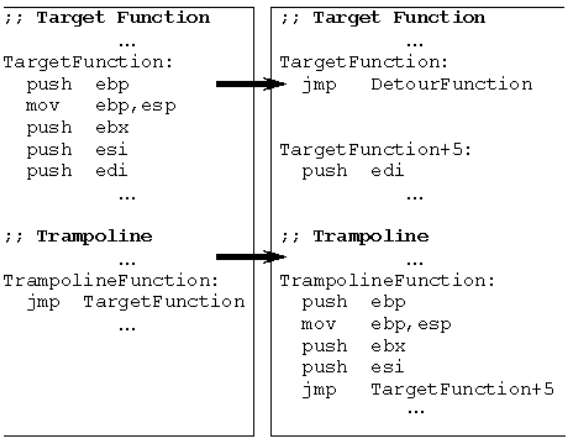
Microsoft Detours biblioteka

Detours yra biblioteka skirta Win32 funkcijoms perimti. Autoriai – Microsoft tyrimų laboratorijos darbuotojai. Veikimo principas: ji pakeičia pačius pirmuosius funkcijos vykdymo baitus į vartotojo sukurtą funkciją, kuri pakeičia originalą. Originalios funkcijos kvietimas apsaugotas su nukreipiančiąja funkcija (angl. *Trampoline*). Iškvietus perimtą funkciją, kvietimas perduodamas į perrašytą jos variantą. Kai atliekamos papildomos operacijos, dėl kurių buvo perimta funkcija, kviečiama originali funkcija. Tuomet ir panaudojamas nukreipiančiosios funkcijos išsaugotos originalios funkcijos kvietimas (tai atliekama dėl to, kad vėl neiškviestume perrašytos funkcijos ir nepapultume į amžiną ciklą). 1 pav. parodoma kaip vyksta perrašytos funkcijos kvietimas ir sąsaja su originaliąja prieš ir po perėmimo (Hunt *et al.* 1999).



1 pav. Kreipimasis prieš ir po perėmimo (Abramov et al, 2008)

Detours biblioteka funkcijas perima perrašydama procesuose esančias jų kvietimo lenteles. Kiekvienai funkcijai biblioteka iš tikrųjų perrašo dvi funkcijas. Viena kviečiama vietoj tikrosios, o kita – aktyvuojama nukreipiančiosios.



2 pav. Batuto ir originalios funkcijos programinis kodas prieš ir po Detours bibliotekos įterpimo (kairė ir dešinė) (Abramov et al, 2008)

2 pav. pateiktas pavyzdys kaip atrodo originalus ir perimtos funkcijos kodas atmintyje (Hunt *et al.* 1999). Visų pirma Detours išskiria atminties nukreipiančiajai funkcijai. Tuomet į jos vietą nukopijuojami mažiausiai 5 baitai originalios funkcijos (tai minimalus kiekis baitų, kai galima atlikti besąlyginę *jmp* komandą). Jei funkcija-taikinys yra trumpesnė nei 5 baitai – Detours biblioteka grąžina klaidą.

Detours biblioteka leidžia perimti bet kokios bibliotekos ir bet kokią funkciją. Versija 32 bitų sistemoms platinama nemokamai (skirta nekomerciniam naudojimui), o 64 bitų – mokama (vienkartinis licencijos mokestis – 10 $).

EasyHook biblioteka

Šis projektas palaiko nekontroliuojamo (angl. *Unmanaged*) programinio kodo perėmimą naudojant kontroliuojamą (angl. *Managed*) kodą. Funkcionalumas pasiekiamas naudojant C# aplinką bei naujas operacines sistemas (Windows 2000 SP4, Windows XP x64, Windows Vista x64 ir Windows Server 2008 x64) (Allaeys. 2009).

EasyHook biblioteka pateikia keletą naujų dalykų. Visų pirma ji užtikrina, kad perimtoje programoje neliktų jokių resursų ir atminties šiukšlių. Biblioteka taip pat leidžia naudoti kontroliuojamas dorokles (angl. *Handler*) skirtas nekontroliuojamoms API funkcijoms perimti. Tai leidžia naudotis daugybe kontroliuojamo kodo suteikiamų galimybių: .NET tolimų procesų bendravimo funkcionalumu (angl. *.NET Remoting*), WCF (funkcionalumas, skirtas kurti, sujungtas, orientuotas į paslaugų programas) ir WPF (vartotojo sąsajos atvaizdavimui). Taip pat svarbu paminėti, kad EasyHook leidžia su ta pačia programa (angl. *Assemblie*) perimti 32 ir 64 bitų procesuose esančias funkcijas naudojant 32 ir 64 bitų procesus (Allaeys. 2009).

EasyHook biblioteka stabilią versiją pasiekė 2009m. kovo 8d. Bet kol kas turi keletą smulkių trūkumų, susijusių su kontroliuojamu kodu, kurie bus ištaisyti kitoje versijoje.

Bibliotekų testavimas

Bibliotekos testuojamos norint išsiaiškinti, kuri biblioteka turi mažiausią poveikį perimtų procesų veikimo spartai. Tai yra labai svarbu, kadangi pasirinkta biblioteka bus naudojama realaus laiko failų nuskaitymo sistemoje. Todėl poveikis turi būti minimalus, tiek kuriant naujus procesus, tiek kviečiant perimtas funkcijas.

Norint nustatyti perimtos programos pradžios pokyčius, visų pirma išmatuojamas jos įprastinės pradžios laikas. Tam pasirinktas Git (versijų kontrolės sistema) teikiamas funkcionalumas – komandinės eilutės komanda *time*. Ji įvykdo pateiktą komandų grandinę (angl. *Pipeline*) ir atspausdina ekrane sugaištą realų bei procesoriaus laiką.

Kiekvienai iš bibliotekų parašytos programos-serveriai. Jie palaiko komandinės eilutės formatą. Kviečiant abi programas nurodoma, ką reikia atlikti ir su kokia taikomąja programa. Jei duodama komanda *run* – nurodyta taikomoji programa yra tik paleidžiama ir laukiama, kol naujas procesas baigs savo darbą. Jei *hook* – į nurodytą taikomąją programą įterpiama viena iš bibliotekų (priklausomai nuo serverio) ir perima funkcijos *CreateFile* kvietimą. 3 pav. pateiktas taikomosios programos, kuri naudojama bibliotekų įsiskverbimo spartai nustatyti, programinis kodas. Kaip matome, programa neatlieka jokių užduočių, tiesiog pradėjusi veikti iškart baigia darbą. Tai užtikrina, kad jos veikimo trukmės nelems vykdomų komandų trukmė.

```
#include <tchar.h>

int _tmain(int argc, _TCHAR* argv[])
{
    return 0;
}
```

3 pav. Taikomosios programos, kuri naudojama bibliotekų įsiskverbimo greičiui nustatyti, programinis kodas

4 pav. pateiktos komandos, kurios naudotos programos pradžios, veikimo ir užsidarymo spartai nustatyti naudojant Detours (Abramov. 2008) biblioteką. Komandos kartojamos 10 kartų, kad būtų gauti kuo įvairesni rezultatai. Identiškos komandos naudojamos EasyHook (Husse. 2008) testavimui, tik vietoje *DetoursHookCenter.exe* naudojama *FileMon.exe* (EasyHook serveris) programa.

```
clear
echo Detours bibliotekos paleidimo testas.
echo Testas bus kartojamas 10 kartu.

for ((  i = 1 ;  i <= 10;  i++  ))
do
    echo
    echo
    echo Bandymas Nr.: $i
    echo RUN
    time DetoursHookCenter.exe run HookDllLoadingTest
    echo
    echo HOOK
    time DetoursHookCenter.exe hook HookDllLoadingTest
done
```

4 pav. Detours bibliotekos įsiskverbimo testavimo komandos

Testuojant perimtų funkcijų veikimo spartos pokyčius naudojama kita taikomoji programa. Ji taip pat 10 kartų atlieka failo kūrimo testą. Pats testas: kuriamas objektas *CFile* su nuoroda, kad failas būtų sukurtas iš naujo. Tokiu būdu iškviečiama *CreateFile* funkcija. Visa tai atliekama 20000 kartų, laikas sumuojamas ir dalinamas iš 20000. Taip gaunamas laikas sugaištas *CFile* objektui sukurti (kartu ir *CreateFile* funkcijai iškviesti). 5 pav. pateiktas šios taikomosios programos programinis kodas.

```
for(int k = 0; k < 10; k++)
{
    time_t start, stop;
    int nCount = 20000;

    time(&start);

    for(int i = 0; i < nCount; ++i)
    {
        CFile file(_T("Test.txt"), CFile::modeCreate);
    }

    time(&stop);

    double dDiff = difftime(stop, start);
    printf("Bendras laikas     %.5f s. \n", dDiff);
    double dOneFile = dDiff / nCount;
    printf("Vieno failo laikas %.5f s. \n\n", dOneFile);
}
```

5 pav. CreateFile kvietimo testas

Funkcijos kvietimo testui taip pat naudoti bibliotekų serveriai su komandomis *run* ir *hook*. 6 pav. pateiktos komandos, naudotos EasyHook bibliotekos testavimui.

```
clear
echo EasyHook bibliotekos veikimo testas.
echo RUN
FileMon.exe run FileUsage.exe
echo
echo HOOK
FileMon.exe hook FileUsage.exe
```

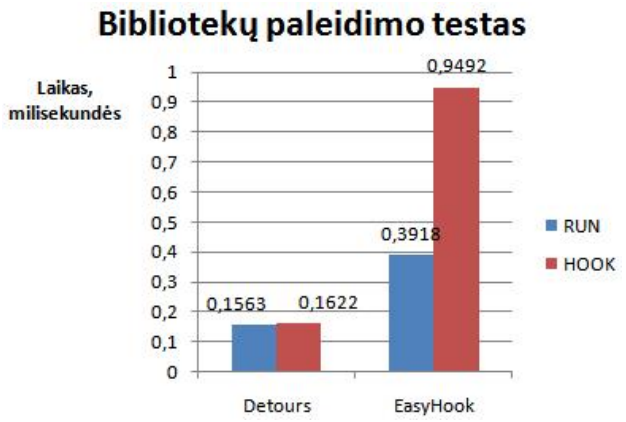
6 pav. EasyHook bibliotekos CreateFile kvietimo testo komandos.

Fig. 6. Batch file for EasyHook library CreateFile call test.

Testavimo rezultatai

Testai atlikti Windows XP sistemoje. Kompiuterio duomenys: Intel(R) T2250 procesorius, 3 GB operatyvioji atmintis.

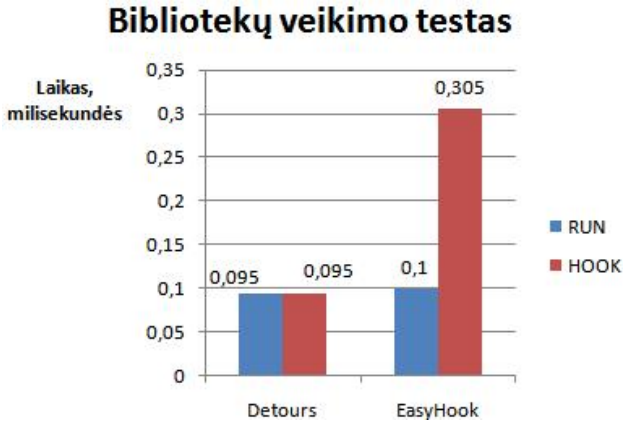
7 pav. pateiktas grafikas, vaizduojantis bibliotekų įtaką pradedančiai veikti programai.



7 pav. Bibliotekų paleidimo testas

Naudojant Detours biblioteką taikomosios programos pradžios trukmė pasikeičia labai nežymiai. Nuo 0,1563 iki 0,1622 milisekundės. Tai įtakoja tik 3,77 % sulėtėjimą. Naudojant EasyHook biblioteką rezultatai skiriasi net keletą kartų. Pradedant be bibliotekos – 0,3918 milisekundės, o startuojant su EasyHook – 0,9492. Tai sudaro 142,27 % sulėtėjimą. Tai reiškia, kad EasyHook biblioteka, programos pradžios trukmę prailgino šiek tiek daugiau nei dvigubai.

8 pav. pateiktas funkcijų veikimas įprastu režimu ir perėmus jas su testuojamomis bibliotekomis.



8 pav. Bibliotekų veikimo testas

Kaip matome, Detours biblioteka neturėjo jokios įtakos funkcijos vykdymo trukmei. O EasyHook biblioteka pateikė kitokius rezultatus. Funkcijos vykdymo trukmė nuo 0,1 milisekundės pailgėjo iki 0,305 milisekundės. Tai sudaro 305 % pradinio laiko arba 3 kartų sulėtėjimą.

Tokių testų rezultatų buvo galima tikėtis, nes EasyHook bibliotekos dokumentacijoje neaptariama jos sparta. Tuo tarpu Detours dokumentacijoje bibliotekos sparta yra pabrėžiama. 9 pav. pateiktas Detours bibliotekos spartos palyginimas su kitais API perėmimo mechanizmais (Hunt *et al.* 1999).

Interception Technique	**Intercepted Function**	
	Empty Function	`CoCreate-Instance`
Direct	0.113µs	14.836µs
Call Replacement	0.143µs	15.193µs
DLL Redirection	0.143µs	15.193µs
Detours Library	0.145µs	15.194µs
Breakpoint Trap	229.564µs	265.851µs

9 pav. Įsiskverbimo metodų palyginimas

Šiame teste buvo palygintas tuščios funkcijos ir *CoCreateInstance* veikimas naudojant įvairias įsiskverbimo technikas. Testas atliktas kompiuteryje su 200 MHz Pentium Pro procesoriumi (Hunt *et al.* 1999). Kaip matyti, Detours naudojimas labai mažai trukmės atžvilgiu skiriasi nuo metodų, kurie pripažinti kaip greičiausi.

Tyrimai šioje srityje

Beieškant informacijos apie minėtas bibliotekas paaiškėjo, jog tyrimai šioje srityje nėra labai plėtojami. Windows API perėmimo metodų ir bibliotekų palyginimų yra labai mažai. Daugiausia - pačių gamintojų pristatomieji straipsniai, apžvelgiantys gerąsias savybes.

Išvados

1. Atlikus metodų apžvalgą, nustatyta, kad perimti darbui su failų sistema skirtas funkcijas gali tik Detours ir EasyHook bibliotekos. Windows API pateikiama SetWindowsHookEx funkcija tinkama tik sistemoje perduodamoms žinutėms ir jų grandinėms perimti.

2. Atlikus EasyHook ir Detours bibliotekų testavimą paaiškėjo, kad EasyHook labai stipriai sulėtina tiek programos pradžią, tiek perimtų funkcijų vykdymo trukmę. Tuo tarpu Detours biblioteka turi labai nežymią arba išvis jokios įtakos minėtiems procesams.

Literatūra

Abramov, A. API Hooking with MS Detours [interaktyvus] 2008. [žiūrėta 2010 m. Kovo 10 d.]. Prieiga per internetą: <http://www.codeproject.com/KB/DLL/funapihook.aspx>.

Allaeys, S. 2009. Design and Implementation of a Portable Virtualization System, *Bachelor Thesis, Helsinki Metropolia University of Applied Sciences*, 2009.

Čeponis, D.; Radvilavičius, L. 2008. Windows API funkcijų stebėjimas, procesų perėmimas ir stebėjimas, 11-osios Lietuvos jaunųjų mokslininkų konferencijos „Mokslas – Lietuvos ateitis“, įvykusios Vilniuje 2008 m. balandžio mėn. 11 d., pranešimų medžiaga.

Hunt, G.; Brubacher, D. 1999. Detours: Binary Interception of Win32 Functions, *USENIX Windows NT Symposium, Seattle*, 1999.

Husse, C. EasyHook - The reinvention of Windows API hooking [interaktyvus] 2008. [žiūrėta 2010 m. Kovo 15 d.]. Prieiga per internetą: <http://www.codeproject.com/KB/DLL/EasyHook64.aspx>.

Detours - Microsoft Research [interaktyvus] 2010. [žiūrėta 2010 m. Kovo 10 d.]. Prieiga per internetą: <http://research.microsoft.com/en-us/projects/detours>.

EasyHook - The reinvention of Windows API Hooking [interaktyvus] 2010. [žiūrėta 2010 m. Kovo 15 d.]. Prieiga per internetą: < http://easyhook.codeplex.com>.

WINDOWS API HOOKING LIBRARIES RESEARCH

D. Čeponis, L. Radvilavičius

Abstract

The paper describes methods how to apply Windows API hooking with third party libraries and solutions. In this research were used Windows API function SetWindowsHookEx, Detours and EasyHook libraries. Libraries methods, features and advantages were discussed in this paper.

The practical part contains libraries tests. In analysis was tested target program start with hooking library and injected function call.

Keywords: Windows API, functions interception, hooking library, research, C++, C#, .NET, Detours, EasyHook, SetWindowsHookEx, Open Source.