VILNIAUS GEDIMINO TECHNIKOS UNIVERSITETAS

Lukas RADVILAVIČIUS

REALIOJO LAIKO SKENAVIMO ANTIVIRUSINIŲ SISTEMŲ NAŠUMO CHARAKTERISTIKŲ TYRIMAS

DAKTARO DISERTACIJA

TECHNOLOGIJOS MOKSLAI,
INFORMATIKOS INŽINERIJA (07T)

VGTU
LEIDYKLA
Vilnius TECHNIKA 2012

Disertacija rengta 2006–2012 metais Vilniaus Gedimino technikos universitete.

Mokslinis vadovas

prof. habil. dr. Antanas ČENYS (Vilniaus Gedimino technikos universitetas, technologijos mokslai, informatikos inžinerija – 07T).

VGTU leidyklos TECHNIKA 2109-M mokslo literatūros knyga
http://leidykla.vgtu.lt

ISBN 978-609-457-423-8

lukas@fmf.vgtu.lt

VILNIUS GEDIMINAS TECHNICAL UNIVERSITY

Lukas RADVILAVIČIUS

REAL-TIME ANTIVIRUS SCANNING METHODS CHARACTERISTICS' EFFICIENCY STUDY

DOCTORAL DISSERTATION

TECHNOLOGICAL SCIENCES,
INFORMATICS ENGINEERING (07T)

VGTU
LEIDYKLA
TECHNIKA
Vilnius 2012

Doctoral dissertation was prepared at Vilnius Gediminas Technical University in 2006–2010.

Scientific Supervisor

Prof Dr Habil Antanas ČENYS (Vilnius Gediminas Technical University, Technological Sciences, Informatics Engineering – 07T).

Reziumė

Disertacijoje nagrinėjamas realiojo laiko antivirusinių sistemų efektyvumo didinimas naudojant taisyklėmis grįstą ekspertinę sistemą. Eksperimentams naudota iš realių vartotojų surinkta informacija apie vykdomų veiksmų sekas su bylų valdymo sistema.

Darbas apima realiojo laiko antivirusinės sistemos projektavimo, kūrimo procesus ir eksperimentinį tyrimą.

Disertaciją sudaro įvadas, trys skyriai, bendrosios išvados, naudotos literatūros ir autoriaus publikacijų disertacijos tema sąrašai.

Įvadiniame skyriuje aptariamas problemos formulavimas, darbo aktualumas, aprašomas tyrimų objektas, formuluojamas darbo tikslas ir uždaviniai, aprašoma tyrimu metodika, darbo mokslinis naujumas, darbo rezultatų praktinė reikšmė ir ginamieji teiginiai.

Pirmajame skyriuje analizuojama kenksmingo programinio kodo raida, jo sandara ir elgsenos pavyzdžiai. Analizuojami realiojo laiko skenavimo mechanizmų realizacijos būdai ir veikimo principai. Aptariama Windows API funkcijų sekų stebėjimo metodologija, analizuojamas jos pritaikymas realiojo laiko antivirusinėms sistemoms kurti. Daug dėmesio skiriama ekspertinių sistemų taikymo informacinėse saugos sistemose būdai ir galimybės. Aptariama kenksmingo programinio kodo techninė analizė, informacijos saugos sistemų projektavimo metodai.

Antrajame skyriuje suformuluojami sistemai keliami funkciniai ir nefunkciniai reikalavimai, pateikiamas sistemos prototipo projektas, kurio svarbiausios dalys atvaizduojamos tiek UML, tiek UMLSec diagramomis.

Trečiajame skyriuje aprašomas sukurtas realiojo laiko skenavimo sistemos prototipas, demonstruojamos jo galimybės, atskleidžiama ekspertinės sistemos struktūra. Daug dėmesio skiriama laboratorinėms sąlygoms, kuriomis buvo vykdomi tyrimai su KPK bei analizuojami vartotojų veiksmai dirbant su kompiuteriniais įrenginiais. Detaliai aptariama virtualios laboratorijos struktūra, aprašomi laboratorijos adaptacijos darbai ir pasiekti rezultatai. Remiantis atliktų tyrimų rezultatais daromos išvados, kad ekspertinių sistemų taikymas, siekiant pagerinti realiojo laiko skenavimo antivirusinės sistemos darbą, yra galimas.

Disertacijos tema paskelbti penki straipsniai: trys straipsniai recenzuojamuose mokslo žurnaluose ir du straipsniai kituose leidiniuose. Disertacijos tema atliktų tyrimų rezultatai pristatyti keturiose mokslinėse konferencijose.

Abstract

The dissertation analyses the issue of an improvement of the real-time anti-virus systems efficiency via the use of a rule-based expert system. Information used for experiments was collected from the real users related to the sequences of actions carried out with the file management system.

The paper includes design and development processes, as well as pilot testing of the real-time anti-virus system.

The dissertation consists of an introduction, three chapters, general conclusions, a list of references and a list of the author's publications on the theme of the thesis.

The introductory chapter discusses the studied issue, the relevance of the work, provides a description of the study subject, formulates an objective and tasks of the paper, describes the study methodology, a scientific novelty of the paper, a practical value of the result of the thesis, as well as its hypotheses.

The first chapter examines the development of malicious software code, its structure and behaviour patterns. The implementation methods and mechanisms for an operation of the real-time scanning methods are analysed. The monitoring methodology for the sequences of Windows API functions is discussed, its application for the creation of real-time anti-virus systems is analysed. Moreover, a considerable attention is paid to the methods and capabilities of the application of expert systems in the information security systems. A technical analysis of the malicious software code, as well as design techniques for the information security systems, is discussed.

The second chapter formulates functional and non-functional requirements of the system, discloses the project of a system prototype, the key parts of which are displayed both by UML and UMLSec diagrams.

The third chapter describes a created real-time scanning system prototype, shows its potential and discloses the structure of an expert system. Moreover, a considerable attention is paid to the laboratory conditions under which the studies on the malware are performed, and the user actions concerning dealing with computer-related devices are analysed. The structure of a virtual laboratory is discussed in detail, the laboratory adaptation tasks and the achieved results are described. Based on the study results, it is concluded that the application of expert systems to improve the scanning of a real-time antivirus system is possible.

Five articles were published on the theme of the dissertation: three articles were published in refereed scientific journals, two articles – in other publications. The results of the studies performed on the theme of the dissertation were presented in four conferences.

Žymėjimai

Santrumpos

KPK – kenksmingas programinis kodas;

AV – antivirusinė sistema;

API – aplikacijų programavimo sąsaja;

RBAC – vaidmenimis pagrįsta prieigos kontrolė;

UML – vieninga modeliavimo kalba;

OS – operacinė sistema;

SegSlice – gaudymo struktūra, matuojanti ir įgyvendinanti ryšius tarp programos kodo ir duomenų elementų, naudojant x86 segmentavimo sistemą;

FS – failų išdėstymo sistema;

FAT – FAT failų išdėstymo sistema;

I/O – įvestis/išvestis;

MSDN – *Microsoft* kūrėjo tinklas;

DLL – dinaminė biblioteka;

PE – vykdomosios bylos formatas;

JMP – procesoriaus instrukcija;

NTFS – failų išdėstymo sistema;

WDK – tvarkyklių rinkinys;

IRP – įeities/išeities užklausos paketas;

I/O Request Packet – įeities/išeities užklausos paketas;

VFS – sąsaja tarp operacinės sistemos branduolio ir realios failų išdėstymo sistemos;

NFS – tinklinė bylų išdėstymo sistema;

FUSE – vartotojo lygio bylų išdėstymo sistema;

Avfs – antivirusinės sistemos bylų išdėstymo sistema;

Oyster – virusų peržiūrėjimo variklis, integruotas Linux branduolyje;

Linux – Linux operacinė sistema;

NetBSD – NetBSD operacinė sistema;

FreeBSD – FreeBSD operacinė sistema;

OpenSolaris – OpenSolaris operacinė sistema;

Mac OS X – Mac OS X operacinė sistema;

ClamFS – vartotojo lygio failų išdėstymo sistema;

Norton AntiVirus – antivirusinė sistema;

COM – bylos tipas;

SOD – vaidmenų atskyrimas;

SOD (SSD) – statinis vaidmenų atskirimas;

SOD (DSD) – dinaminis vaidmenų atskirimas;

ClamAV – ClamAV antivirusinė sistema;

Main signature DB – duomenų bazė iš ClamAV – didžiausia KPK duomenų bazė iš šio gamintojo;

Daily signature DB – ClamAV duomenų bazė, kurioje saugomi naujausi KPK parašai;

File extension DB – vidinė duomenų bazė, kurioje saugomi skenuotinų ir neskenuotinų failų trumpiniai;

Standard file DB – duomenų bazė, kuri saugo standartinių operacinės sistemos failų kontrolines sumas, kurie pažymėti kaip patikimi ir niekuomet neturėtų būti skenuojami;

Internal hash DB – duomenų bazė, kurioje saugomos visų skenuotų failų kontrolinės sumos, kad antivirusinei programai nereikėtų to paties darbo daryti kelis sykius;

Most used files DB – daugiausia naudojamų bylų duomenų bazė;

DHCP serveris – dinaminių adresų dalinimo serveris;

HTML – penktos versijos hiperteksto ženklinimo kalba;

HTML5 Canvas – penktos versijos hiperteksto ženklinimo kalbos elementas;
QEMU – operacinės sistemos virtualizacijos serveris;
XML-RPC – protokolo pavadinimas;
SSL/TLS – šifruotas protokolas;
VPN – virtualus privatus tinklas;
TCP/IP – TCP/IP interneto protokolas;
FTP – failų persiuntimo protokolas;
HTTP – hipertekstų persiuntimo protokolas;
OpenSWAN – virtualaus privataus tinklo serveris;
RDBVS – releacinės duomenų bazių valdymo sistema.

Turinys

Contents

Įvadas

Problemos formulavimas

Informacijos saugos, t. y. konfidencialumo, vientisumo ir prieinamumo, užtikrinimo klausimas asmeniniame kompiuteryje ar mobiliuosiuose įrenginiuose išlieka aktualus nepaisant to, kad jį bandoma išspręsti nuo pat asmeninio kompiuterio atsiradimo. Sprendžiant šį uždavinį svarbu užtikrinti darną tarp sistemos našumo ir naudotojų darbo patogumo bei duomenų saugos lygio. Antivirusinės sistemos yra kritiškai svarbios apsaugai nuo kenksmingo programinio kodo (KPK), bet kartu pasižymi vis didėjančiu poreikiu kompiuterio resursams, kuris negali būti kompensuotas didėjančia kompiuterių sparta. Antivirusinė, komercinė ar atvirojo kodo programa reguliariai arba pagal poreikį tikrina skirtingus naudotojo kompiuterio nustatymus, procesus ir duomenis. Tikrinant naudojami kompiuterio resursai – procesorius, operatyvioji atmintis. Didėjantį antivirusių sistemų poreikį kompiuterio resursams lemia šiuo metu dominuojanti KPK aptikimo technologija, paremta KPK parašais (didėjant parašų bazei ilgiau trunka skenuojamų bylų analizė), taip pat sudėtingėjantis operacinių sistemų funkcionalumus. Jis lemia, kad vis daugiau bylų ir sisteminių kreipinių turi būti patikrinta antivirusinės sistemos. Didėjanti konkurencija tarp vartotojo taikomosios programinės įrangos ir saugos užtikrinimo antivirusinės įrangos dėl kompiuterio resursų, kurių kritiškiausiu gali būti laikomas procesoriaus laikas, verčia ieškoti

būdų, kaip sumažinti saugos sistemos poveikį vartotojų darbo našumui nemažinant saugos lygio.

Antivirusinių sistemų poreikio sistemos resursams minimizavimo uždavinį bandoma spręsti mažinant kreipimųsi į antivirusinę sistemą skaičių. Tokie algoritmai realizuoti daugelyje komercinių antivirusinių sistemų („Kaspersky“, „Symantec“ ir t. t.), tačiau sprendimų priėmimo mechanizmų (kas turi būti skenuojama, o kas ne) veikimo principai yra patentuoti ir neviešinami. Tai stabdo nekomercinių antivirusinių sistemų pritaikymą ir sukelia poreikį atlikti nepriklausomus mokslinius tyrimus šioje srityje.

Rengiant disertaciją buvo ieškoma metodų ir būdų, kurių rezultatas – naujoviška aptikimo technologija, grindžiama mažų įrenginio resursų poreikiu, būtų puikiai pritaikoma mobiliesiems telefonams ir planšetiniams kompiuteriams. Atlikus analizę ir eksperimentinius tyrimus su realia kenksminga programine įranga, gauti realiojo laiko antivirusinių sistemų našumo charakteristikų tyrimų rezultatai.

Tiriamoji problema – naudotojo kompiuterio darbo našumo išlaikymas tuo metu, kai kompiuteryje veikia antivirusinė programa.

Darbo aktualumas

Disertacijoje tiriami realiojo laiko antivirusinių programų optimizavimo klausimai. Didėjantis kompiuterinių resursų poreikis iš antivirusinių sistemų pusės lemia antivirusinių sistemų taikymo mažo galingumo kompiuterinėse sistemose (pvz., mobilieji įrenginiai, planšetiniai ir tinkliniai kompiuteriai, integruotos sistemos) pritaikymą, mažesnį vartotojų darbo našumą, žemą darbo komforto lygį ir net gali paskatinti atsisakyti antivirusinės sistemos. Tai gali turėti neigiamų pasekmių asmeninio kompiuterio saugumui. Šiuolaikiniai KPK aptikimo metodai, paremti anomalijų analize, negali užtikrinti tokio pat patikimumo lygio, kaip parašais paremti metodai, todėl aktualus išlieka pastarųjų metodų taikymo optimizavimo klausimas. Egzistuojančių optimizavimo sprendimų komercinėse antivirusinėse sistemose, kurių veikimo principai nėra viešinami, našumo charakteristikos neleidžia teigti, kad uždavinys yra išspręstas. Taip pat egzistuoja tų analogiškų optimizavimo sprendimų poreikis, kurie nėra apsaugoti patentais ir galėtų būti naudojami nekomercinėse bei mokslinėse sistemose. Jei į antivirusinių sistemų našumo gerinimo užduotį nebus gilinamasi ir ši sritis nebus tiriama, dėl didėjančių duomenų srautų antivirusinių realiojo laiko skenavimo sistemas naudoti asmeniniuose įrenginiuose greitai taps sunkiai įmanoma. Tyrimų šioje srityje aktualumą taip pat pagrindžia daugelis publikacijų moksliniuose žurnaluose, kurių autoriai akcentuoja šiame darbe nagrinėjamos temos aktualumą.

Tyrimo objektas

Disertacijos tyrimo objektas – realiojo laiko antivirusinių programų darbo efektyvumo tyrimas.

Darbo tikslas

Metodo, padidinančio realiojo laiko skenavimo proceso efektyvumą ir greitaveiką, sukūrimas.

Darbo uždaviniai

Darbo tikslui pasiekti reikia išspręsti šiuos uždavinius:

1. Ištirti esamus realiojo laiko failų išdėstymo sistemos skenavimo mechanizmus ir jų taikymo galimybes: aptikimo metodus, taikymo problemas, patentus, failų išdėstymo sistemos stebėjimų metodus. Atlikti techninę KPK analizę.
2. Pasirinkti informacijos saugos sistemų projektavimo metodą realiojo laiko skenavimo sistemos architektūrai kurti.
3. Sukurti realiojo laiko skenavimo sistemos funkcinius ekspertinės sistemos reikalavimus ir sistemos projektą.
4. Eksperimentiškai nustatyti ir aprašyti tipinę operacinės sistemos darbo su failų sistema veiksmų seką nustatytam laiko intervalui.
5. Remiantis projektu sukurti eksperimentinę sistemą.
6. Eksperimentiškai įvertinti pasiūlyto metodo efektyvumą.

Tyrimo metodika

Darbo tikslui pasiekti taikomi šie tyrimo metodai:

- Lyginamosios analizės ir literatūros analizės metodai buvo taikomi analizuojant antivirusinių sistemų realiojo laiko mechanizmų principus, esamų kenkėjiškų programų veikimo metodus.
- Apibendrinimas – analizės ir tyrimų rezultatų susisteminimas bei reikšmingumo nustatymas.
- Eksperimentinio tyrimo metodai buvo taikomi nagrinėjant KPK veikimą vartotojų asmeniniuose kompiuteriuose, vykdant metodo efektyvumo bandymus.

Darbo mokslinis naujumas

Rengiant disertaciją, gauti šie informatikos inžinerijos mokslui nauji rezultatai:

1. Pasiūlyta nauja ekspertinė sistema, skirta atvirojo kodo antivirusinių sistemų našumo charakteristikų didinimui.
2. KPK tyrimams pritaikyta virtualizuota saugumo laboratorija su galimybėmis testuoti KPK lokaliose ir nutolusiose tarnybinėse stotyse.

Darbo rezultatų praktinė reikšmė

Šios srities tyrimai padeda prisitaikyti prie didėjančių duomenų kiekių ir jų tikrinimo realiojo laiko skenavimo priemonėmis. Tyrimų metu buvo surinkti duomenys apie vartotojų darbo metu atliekamus veiksmus ir jų sekas su bylų valdymo sistema, sudaryta ekspertinė sistema, leidžianti pagerinti realiojo laiko skenavimo sistemų darbą. Eksperimentiniai metodo bandymų rezultatai leidžia teigti, kad metodo taikymas leidžia sumažinti antivirusinių sistemų poreikį kompiuteriniams resursams, padidinti kompiuterių naudotojų darbo našumą ir kokybę, pagerinti nekomercinių, atvirojo kodo antivirusinių sistemų kokybę.

Gauti tyrimų rezultatai gali būti naudojami kuriant mažiau resursų reikalaujančias antivirusines programas kiekvienam šiuolaikiniam kompiuteriui, įskaitant ribotus resursus turinčius planšetinius ir integruotus įrenginius.

Ginamieji teiginiai

Remiantis tyrimų rezultatais suformuluoti šie ginamieji teiginiai:

1. Ekspertinės sistemos, atliekančios išankstinę skenuojamų failų analizę, leidžia padidinti AV našumą be poveikio antivirusinės sistemos patikimumui.
2. Failų išdėstymo sistemos tvarkyklių taikymas yra efektyvesnis už operacinės sistemos funkcijų perėmimą (angl. *hooking*), norint realiuoju laiku stebėti veiksmus, atliekamus su failų išdėstymo sistema.
3. Reliacinė duomenų bazė gali būti naudojama ekspertinių sistemų taisyklėms saugoti ir interpretuoti.

Darbo rezultatų aprobavimas

Disertacijos tema išspausdintos keturios mokslinės publikacijos: trys straipsniai publikuoti tarptautinėse duomenų bazėse referuojamuose žurnaluose, vienas – konferencijos leidinyje.

Disertacijoje atliktų tyrimų rezultatai paskelbti keturiuose mokslinėse konferencijose:

- 16-ojoje Lietuvos jaunųjų mokslininkų konferencijoje *„Mokslas – Lietuvos ateitis“* 2012 m. Vilniuje, Lietuvoje.
- Tartautinėje konferencijoje *„6th International Conference on Internet Technology and Secured Transactions (ICITST-2011)“* 2011 m. Abu Dabyje, Jungtiniuose Arabų Emyratuose.
- 12-ojoje Lietuvos jaunųjų mokslininkų konferencijoje *„Mokslas – Lietuvos ateitis“* 2009 m. Vilniuje, Lietuvoje.
- 11-ojoje Lietuvos jaunųjų mokslininkų konferencijoje *„Mokslas – Lietuvos ateitis“* 2008 m. Vilniuje, Lietuvoje.

Disertacijos struktūra

Diseratciją sudaro įvadas 3 skyriai, bendrosios išvados ir rekomendacijos, literatūros sąrašas ir publikacijų sąrašas.

Darbo apimtis – 110 puslapių, įskaitant priedą, tekste panaudota 40 paveikslų ir 9 lentelės. Rašant disertaciją naudotasi 74 literatūros šaltiniais.

1

Esamų antivirusinių sistemų realiojo laiko skenavimo mechanizmų analizė

Šiame skyriuje analizuojama kenksmingo programinio kodo raida, jo sandara ir elgsenos pavyzdžiai. Analizuojami realiojo laiko skenavimo mechanizmų realizacijos būdai ir veikimo principai. Aptariama Windows API funkcijų sekų stebėjimo metodologija, analizuojamas jos pritaikymas realiojo laiko antivirusinėms sistemoms kurti.

Skyriaus pabaigoje analizuojami ekspertinių sistemų taikymo informacinėse saugos sistemose būdai ir galimybės. Aptariama kenksmingo programinio kodo techninė analizė, informacijos saugos sistemų projektavimo metodai, t. y. vaidmenimis pagrįsta prieigos kontrolė (angl. Role-based access control, RBAC) ir du UML plėtiniai – SecureUML ir UMLsec. Aprašomos abiejų metodikų savybės ir palyginamos modeliuojant specifinę sistemą, ypatingą dėmesį kreipiant į kuriamos sistemos saugą.

Skyriaus tematika paskelbtas vienas autoriaus straipsnis (Radvilavičius *et al.* 2012).

1.1. Kenksmingo programinio kodo raida

Žvelgiant į praeitį teigiama, kad pirmieji virusai pasirodė šeštojo dešimtmečio pabaigoje „Univac1108“ ir „IBM-360/370“ kompiuteriuose. Terminą „virusas“ pirmąsyk paminėjo amerikietis Fredas Koenas (Fred Cohen) 1983 m. 7-ojoje informacijos saugumo konferencijoje (JAV). Šiuo metu kompiuteriniai virusai ir kitas KPK yra vertinami kaip viena didžiausių grėsmių informacinei visuomenei. Duominuojančia apsaugos nuo KPK technologija išlieka antivirusinės sistemos, kurios taikymo efektyvumas labai priklauso nuo KPK parašų bazės atnaujimų valdymo ir nepertraukiamo naudojimo. Vis didėjantis KPK kiekis neigiamai įtakoja antivirusinių sistemų greitaveiką, todėl šiuolaikiniam informacinių technologijų ir komunikacijų sektoriui reikia sprendimų, kurie būtų nematomi, veiktų foniniam rėžime ir neeikvotų resursų, įtakojančių naudotojo darbo efektyvumą., Du svarbiausi veiksniai, lemiantys galutinio naudotojo apsisprendimą naudoti ar nenaudoti antivisurinės sistemos yra – našumas ir antivirusinės programinės įrangos gebėjimas greitai bei korektiškai atskirti infekuotas bylas nuo neužkrėstų. Antivirusinės programos naudoja du KPK paieškos tipus- paieška pagal poreikį, tai yra vartotojo inicijuota užklausa ieškoti KPK kompiuteryje arba jo dalyje, bei automatinė paieška, kitaip vadinami realiojo laiko, fonine apsauga, gyventojo skydu arba autoprotekcija. Pastarasis būdas stebi duomenis realiuoju laiku, t. y. kai duomenys patenka į kompiuterį, kai bylos atidaromos arba atliekami kiti veiksmai operacinės sistemos lygyje. Tuomet, kai aptinkamas kenkėjiškas veiksmas, antivirusinė sistema gali jį užblokuoti iki jam pakenkiant kompiuterio sistemai. Kenksmingas programinis kodas taip pat itakoja sparčiai besivystančią mobiliųjų technologijų sritį. Telefonas tapo ne vien pokalbių įrankiu, bet ir neatsiejama gyvenimo dalimi. Jame dažnai laikoma jautri informacija, telefono saugos mechanizmų pažeidimas gali lemti tiesioginius finansinius nuostolius jo savininkui, būtent dėl to telefonas tapo taikiniu kenkėjiškai programinei įrangai ir programišiams. Atsižvelgiant į šiuolaikinių mobiliųjų įrenginių atminties dydį ir procesoriaus pajėgumą, antivirusinė programinė įranga privalo būti optimizuota darbui su mažais resursų kiekiais.

Pasaulyje naudojami realiojo laiko skenavimo sprendimai yra dviejų tipų (nevieši, t. y. komercializuoti) ir bandomieji-moksliniiai. Jų rezultatai dažniausiai pristatomi moksliniuose leidiniuose, tiesa, vos sėkmingesni yra patentuojami dėl jau minėtos didžiulės konkurencijos.

1.2. Antivirusinės sistemos

Kompiuterijos pradžioje antivirusės sistemos buvo naudojamos apsaugai tik nuo kompiuterinių virusų, o dabartiniai antivirusai apsaugo nuo KPK įvairovės, tad pagrindiniu analizės objektu tampa KPK paieškos mechanizmai.

Dabartinių antivirusų tipų klasifikacija grindžiama funkcionalumu (1.1 lentelė).

1.1 lentelė. Antivirusinių sistemų klasifikacija pagal funkcionalumą
Table 1.1. Anti-virus systems, classification in terms of functionality

Nr.	Tipas	Funkcionalumas
1.	Skeneriai	Grindžiama parašu ir euristiniu metodu
2.	Revizoriai	Fiksuoja FS (failų išdėstymo sistemos) būseną
3.	Monitoriai	Stebi potencialiai pavojingus veiksmus
4.	Vakcinos	Priverčia virusą manyti, kad byla jau užkrėsta

Atsižvelgiant į lentelėje pateikiamas funkcionalumo rūšis, antiviruso veiksmai galėtų būti traktuojami kaip konkreti apsauga ar užkrėsto dokumento įšaldymas, neveiksnumo suteikimas:

1. Infekuotos bylos šalinimas.
2. Prieigos prie infekuoto failo blokavimas.
3. Karantinas (vykdymo blokavimas).
4. Viruso šalinimas iš dokumento kūno – gydymas.
5. Vieno iš veiksmų vykdymas perkrovus kompiuterį.

Struktūrizuoti metodai detalizuojami.

Kenksmingas programinis kodas dažniausiai aptinkamas pasinaudojant vienu iš toliau aptariamų aptikimo metodų:

1. Pagrįstas parašais. Šio metodo pagrindas – iš KPK dalių sudaryta eilutė, pagal kurią atliekama paieška kenksmingame programiniame kode. Parašai daugiausia kuriami rankomis. Šio metodo teigiamos savybės – tikslumas, patikimumas, plačiausiai naudojama patikrinta technologija; neigiamos – nemato naujų polimorfinių KPK, vėluoja atnaujinimai, didėja parašų bazės ir skenavimo greitis, taip pat daug rankų darbo kuriant parašus.
2. Euristinis. Metodas skirtas aptikti KPK, kai parašas sutampa ne 100 %. Šis metodas tinka morfiniams virusams aptikti, bet nėra visiškai patikimas. Taip pat *false-positive* ir nėra gydymo galimybės. Pavyzdžiui:

- stebimi veiksmai naudojami ribotai (FDISK, disko skaidinių skaidymas);
- viruso veikimo emuliavimas;
- dekompiliavimas ir procentinė sutapimų analizė.

3. Anomalijų analizė. Metodas, kurį taikant stebimas procesas, srautas, naudotojo veiksmai (pavyzdžiui, ar procesas nemodifikuoja *exe* bylų). Teigiamos šio metodo savybės: aptinka naują KPK, metodas dinaminis ir prisitaikantis; neigiamos: nepatikimas, didelis *false-positive* skaičius, dalį stebimų įtartinų veiksmų atlieka legalios programos.
4. „Smėlio dėžė“. Tai virtuali mašina, kurioje įvykdomas KPK. Po įtariamo KPK įvykdymo analizuojami sistemai padaryti pakeitimai. Teigiamos savybės: efektyvumas, tinka profesionaliai analizei; neigiamos: didelės kompiuterio resurso laiko sąnaudos.
5. „Baltas sąrašas“. Šiuo metodu leidžiamos tik programos, esančios sudarytame „baltajame sąraše“. Teigiamos savybės: nereikia atnaujinti parašų, efektyvu, tinka didelėms korporacijoms su PĮ naudojimo politika; neigiamos: nėra labai lanksti ir patogi vartotojams, reikia administratoriaus darbo laiko resursų.

1.3. Kenksmingo programinio kodo techninė analizė

Kenksmingos programinės įrangos elgsenos charakterizavimas nėra triviali tyrimo problema ir jau daugybę kartų tokio tipo uždavinius buvo mėginta spręsti. Didelė tokių darbų dalis naudoja kenksmingos programinės įrangos kontrolės srauto šablonus kaip kad instrukcijų ar sisteminių iškvietimų sekos kenksmingai programinei įrangai aptikti (Balzarotti *et al.* 2010; Bayer *et al.* 2009; Christodorescu *et al.* 2003; Kolbitsch *et al.* 2009; Kruegel *et al.* 2004). Atsakydami į tai, kenksmingos programinės įrangos gamintojai dažnai naudoja įvairias maskavimo technologijas, siekdami suklaidinti kenksmingo kodo analizatorius (Christodorescu, Jha 2003; Collberg *et al.* 1998; Sharif *et al.* 2009; Wang *et al.* 2001). Tokie būdai susiduria su problemomis, kylančiomis iš vykdymo dinamikos, kaip kad dinaminiai kodo keliai ir kitų sistemos komponentų (pvz., tinklo latencijos ir signalų) poveikio, kuris gali nulemti kitimus aprašomo kenksmingo programinio kodo šablonuose. Situacija sudėtingesnė branduolio erdvėje, nes operacinių sistemų (OS) branduoliai turi labai dinamišką darbo krūvį, įskaitant pertraukimus, vartotojo procesų koordinavimą ir žemo lygio resursų (pvz., blokų lentelių) valdymas.

Kenksmingų programų, veikiančių branduolyje, aptikimui ir prevencijai yra kita darbų grupė, paremta kodo integralumo tikrinimu (Riley *et al.* 2008; Seshadriet *et al.* 2007). Šis metodas leidžia vykdyti tik autorizuotą kodą ir bet kokį kitą kodą už baltojo sąrašo ribų laiko esant kenksmingą. Todėl šis metodas yra efektyvus branduolio įsilaužėlių paketams, kurie į branduolio erdvę įterpia naują kodą. Tačiau kitos pažengusios tokio tipo kenksmingos programos atlieka atakas, išnaudodamos tik leidžiamą branduolio kodą (t. y. atminties įrenginių iš-

naudojimas (Phrack Magazin), branduolio klaidos ir į grįžtamumą orientuotas programavimas (Hund *et al.* 2009), o su tokiomis atakomis šis metodas nesusidoroja tinkamai. Be to, toks būdas autorizuoja branduolio tvarkyklių kodą, remiantis politika pasitikėti OS kūrėjais ar prekeiviais be sistemiško kodo patikrinimo. Pvz., kodo integralumu paremti metodai (Riley *et al.* 2008; Seshadri *et al.* 2007) leidžia branduolio tekstą ir sąrašą nekenksmingų branduolio modulių, įtrauktų į OS distribucijas. Tokia politika neapsaugo nuo paslėpto kenksmingo kodo autorizuoto kodo viduje. Todėl branduolio tvarkyklių patikros nuo potencialiai kenksmingos elgsenos, nepaisant tokio tipo taisyklių, galimybė yra pageidautina.

Disertacija rengta naudojantis naujausiomis technologijomis, čia analizuojamas metodas, charakterizuojantis branduolio kenksmingo kodo elgseną, naudodamas jo duomenų prieigos šablonus. Kai branduolyje esanti kenksminga programa prisiliečia prie pagrindinių branduolio duomenų, egzistuoja unikalūs branduolio duomenų prieigos šablonai. Todėl galima paimti duomenų prieigos šablonų, nuolat atsirandančių atskirais branduolio vykdymo atvejais tik tuomet, kai aktyvi kenksminga programinė įranga, poaibį ir jį naudojant sugeneruoti kenksmingos programos parašą. Šiuose šablonuose nėra nei kenksmingų programų laikinos kontrolės srauto informacijos, nei kodui specifinės informacijos apie kenksmingą programą. Todėl toks metodas ne taip lengvai apgaunamas ir efektyvesnis tikrinant kenksmingų programų variantus.

Problemos ir sprendimai. KPK aptikimo metodas (kuriuo remiantis rašoma metodinė disertacijos dalis) charakterizuoja kenksmingų programų elgesį naudodamas atminties naudojimo šablonus, išskirtinai stebimus vykdant kenksmingas programas. Jis lygina dviejų tipų branduolio vykdymo atvejus tokiai informacijai apskaičiuoti ir nereikalauja specifinių žinių apie kenksmingą programą. Tai gėrybinis branduolio veikimas, o piktybinis – su kenksminga programa. Tokia metodika susiduria su keletu problemų:

- *Variacijos branduolio veikimo meto elgsenoje***.** Apskritai pilnos branduolio vykdymo kelių aibės gavimas yra gerai žinoma dinaminiu vykdymu grindžiamos metodikos problema. Jei koncentruojamės į duomenų elgseną gėrybinio vykdymo metu, tai problema, nes veikimo metu branduolio elgsena yra itin dinaminė skirtingais vykdymo etapais. Tačiau pagrindinis dėmesys skiriamas branduolio kenksmingos programinės įrangos duomenų elgsenai, kuri nuolat pastebima esant aktyviam kenksmingam kodui. Toks elgesys – uždara kenksmingų veiksmų aibė, tad naudojame keletą kenksmingų branduolio vykdymo atvejų, fiksuodami kenksmingos elgsenos poaibį, kuris atitinka tokius kriterijus.
- *Nereguliarūs prieigos šablonai branduolio dėkluose.* Branduolio dėklai – objektai, turintys nereguliarios prieigos šablonus. Kai iškviečiama ar grąžinama branduolio funkcija, dėklas prieinamas įvairiems tikslams,

kaip kad grąžinamos reikšmės, funkcijų argumentai ir vietiniai kintamieji. Kadangi branduolio kontrolės srautas itin dinamiškas, kodo vietų, kurios naudojasi dėklų ir naudojamų ofsetų dėklų, aibės labai varijuoja. Taip pat branduolio dėklų turinys nėra reguliarus skirtingo vykdymo metu. Paprasčiausias būdas išspręsti šią problemą – pašalinti dėklus iš analizės. Branduolio atminties žymėtojas suteikia identifikatorių branduolio dėklams, ir mes tiesiog pašaliname informaciją apie tokius dinaminius objektus iš savo analizės.

- *Skirtingi ofsetai masyvuose.* Kai kurios duomenų struktūros (pvz., ofsetai ir buferiai) turi ruožą atminties, kurios dalį gali panaudoti vykdymo metu. Pvz., naudojami buferio ofsetai gali būti skirtingi priklausomai nuo to, kokių juose yra duomenų. Ši problema sprendžiama keliais branduolio vykdymo atvejais. Jei naudojamas atminties ofsetas yra skirtingas kiekvieno vykdymo metu, jis nenaudojamas kenksmingos programos parašui, nes jo nebus įmanoma naudoti kito vykdymo metu. Tik duomenų elgsena, atsirandanti nuolatiniais šablonais, kai kenksmingos programos yra aktyvios, tampa kandidatu parašui.

Branduolio kenksmingų programų apsauga, paremta kodo integralumu. Metodai, paremti kodo integralumu (Seshadri *et al.* 2007; Riley *et al.* 2008), leidžia tik autorizuotą branduolio kodą: branduolio tekstą ir branduolio modulius leidžiamame sąraše. Toks metodas efektyvus apsaugant nuo branduolio įsilaužėlių kodų, kurie įterpia savo kodą. Tačiau pažengę įsilaužėlių kodai veikia išvengdami aiškaus kenksmingo kodo įterpimo, naudodami technologijas kaip kad branduolio atminties įrenginiai, branduolio klaidos ar grįžtamasis programavimas, nuo kurių toks metodas apsaugoti negali. DataGene pristato naują požiūrio tašką ir aptinka įsilaužėlių paketus, remdamasis unikalia jų duomenų elgsena. Todėl toks metodas gali būti pritaikytas tokiems problemiškiems įsilaužėlių kodams.

Branduolio įsilaužėlių paketų profiliatoriai. Branduolio įsilaužėlių paketų profiliatoriai (Riley *et al.* 2009; Xuan *et al.* 2009) pateikia įvairius įsilaužėlių kodų elgsenos aspektus, analizuodami įsilaužėlių kodo veiksmus ir tikrindami įtaką vartotojo erdvei. Tokių metodų profiliavimo rezultatas yra specifinis analizuojamai kenksmingai programinei įrangai. DataGene, priešingai, naudoja kenksmingų programų atminties prieigos šablonus, kurių kodo informacija yra generalizuojama. Todėl yra galimybė aptikti įsilaužėlių kodo variantus, kurie panašūs savo elgsena su duomenimis. Taip pat Datagene tiria bendras daugybės įsilaužėlių paketų charakteristikas.

Parašai, paremti duomenų struktūromis. Laika (Cozzie *et al.* 2011) gali nustatyti duomenų struktūras ir klasifikuoti unikalius jų šablonus kenksmingoms programoms. Toks metodas efektyvus vartotojo erdvės kenksmingoms programoms (pvz., botų tinklų programoms), kurios turi savo nuosavą atminties erdvę.

Tačiau branduolio kenksmingų programų kodas ir duomenys yra branduolio atmintyje drauge su leidžiamu branduolio kodu ir duomenimis. Branduolio kenksmingų programų taikinys – leidžiamieji branduolio duomenys ir branduolio kreipinių perėmimas, siekiant panaudoti savo duomenis. Todėl duomenų elgsena branduolio erdvėje yra branduolio ir branduolio kenksmingų programų veiksmų bendras rezultatas.

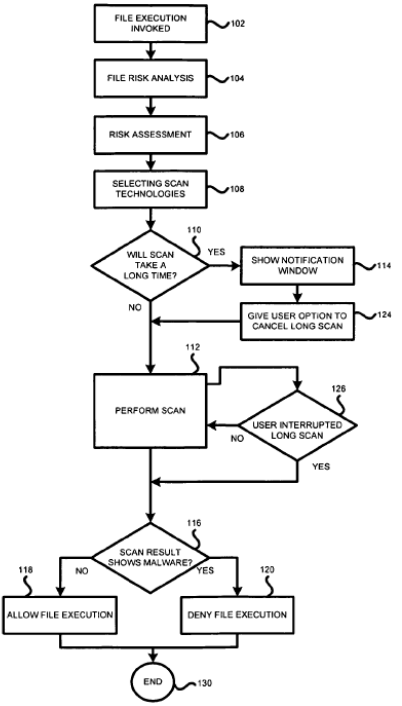
Keli metodai (Lin *et al.* 2011; Dolan-Gavitt *et al.* 2009; Carbone *et al.* 2009) gali aptikti branduolio duomenų struktūras, remiantis duomenų savybėmis, kaip kad duomenų reikšmės ir rodyklių sąsajos. Remiantis duomenų struktūrų aptikimu, tokie metodai taip pat gali aptikti branduolio įsilaužėlio paketus, kurie slepia branduolio duomenų struktūras. Tokių metodų parašai yra duomenų struktūros savybės, o DataGene parašai – kenksmingų programų savybės. Jie generuojami naudojant unikalius kenksmingų programų duomenų prieigos šablonus.

Tarpinis ryšys tarp kodo ir duomenų. SegSlice (Bratus *et al.* 2010) yra gaudymo struktūra, matuojanti ir įgyvendinanti ryšius tarp programos kodo ir duomenų elementų, naudojant x86 segmentavimo sistemą. Šie ryšiai yra apibrėžti programuotojo, naudojančio SegSlice API. Branduolio duomenų šablonai, sugaunami DataGene, atspindi ryšius tarp kodo ir duomenų parodant, kuris kodas greičiausiai mėgins prieiti prie kokių duomenų tipų. Tokie prieigos šablonai sistemiškai sugaunami esant dinaminiam operacinės sistemos branduolio vykdymui, naudojant vizualizacijos technologiją.

1.4. Patentuotos realiojo laiko antivirusinės programinės įrangos skenavimo technologijos

Egzistuoja daugybė realiojo laiko antivirusinės programinės įrangos skenavimo technologijų, kurios yra patentuotos. Kadangi nėra įmanoma aiškiai ir korektiškai aprašyti metodų, naudojamų komercinių antivirusinių sistemų, šiame skyriuje peržvelgiami kai kurie svarbūs patentai, kurie pateikia naujoviškus realiojo laiko stebėjimo ir nuskaitymo, ieškant kenksmingo kodo, būdus.

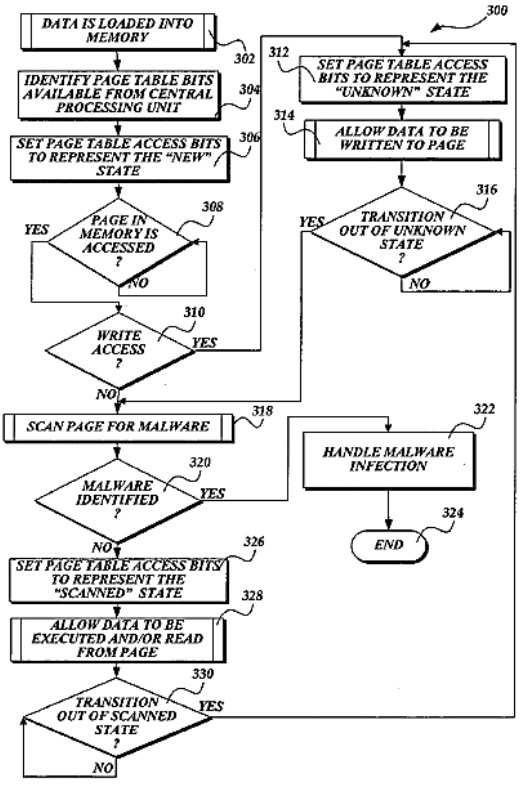
JAV patente (Method and system for antimalware… 2010) Kaspersky Lab patentavo metodiką ir sistemą antikenkėjiškam programinės įrangos nuskaitymui. Išradimas, registruotas 2010 m., pateikia sprendimą vykdomiesiems failams nuskaityti ir kenksmingo programinio kodo paieškai. Išradimo blokinė schema pavaizduota 1.1 pav. Autoriai teigia, kad šis išradimas leidžia sumažinti nuskaitymo laiką, balansuojant greitus (tačiau dažniausia ne tokius išsamius) patikrinimus su išsamiais, tačiau lėtesniais. Skenavimo užklausa turi pereiti nemažai procesų, ir tik paskui sistemai suteikiamas prieigos prie jo leidimas. Dideli failai vertinami atskirai.



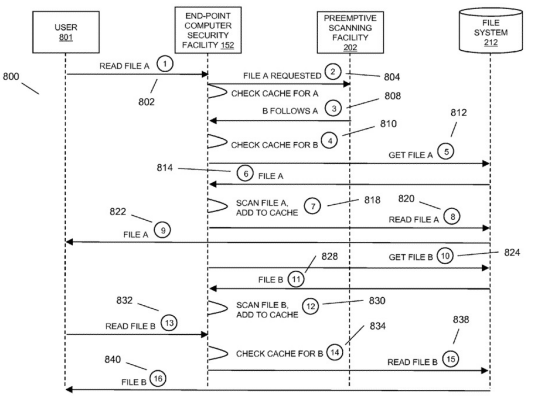
1.1 pav. Patento schema (Method and system for antimalware… 2010)
Fig. 1.1. Patent scheme (Method and system for antimalware… 2010)

Microsoft patente (On-access scan of… 2010) publikuojama kenksmingo kodo, įkrauto į atmintį, identifikacija. Programinės įrangos paprogramės, įgyvendintos šiame išradime, seka puslapių, kurie įkrauti į atmintį, būseną. Patento schema pavaizduota 1.2 pav. Šis išradimas geba aptikti kenksmingą kodą nepaisant to, kokiu būdu jis gavo prieigą prie įrenginio (išvengė aptikimo naudojant šifravimą, išnaudojo programą, kuri jau buvo atmintyje, ar pan.). Pagal specifikacijas ši technologija siūlo galimybę gerinti antivirusinės programinės įrangos našumą. Visos programos, įkrautos į atmintį, laikomos nesaugiomis ir potencialiai kenksmingomis. Todėl sistema kreipiasi į skenavimo variklį infekcijų paieškai, kol jos įvykdomos. Taip pat šis metodas geba aptikti nežinomą kenksmingą programinį kodą remiantis euristiką skenuojant atmintį, kol kenksmingi veiksmai įvykdomi.

Patente (Method and system for preemptive… 2010) pristatoma galimybė leisti sumažinti failų prieigos laiką realiuoju laiku skenuojant nuspėjamuoju pirminiu būdu. Išradimas buvo registruotas 2010 m. Nuspėjamasis nuskaitymas įmanomas dėl failų prieigos našumo įkainių žymėjimo failų išdėstymo sistemoje. Tai gali būti vystoma stebint laiką, kurį užima failų nuskaitymas, ir t. t.



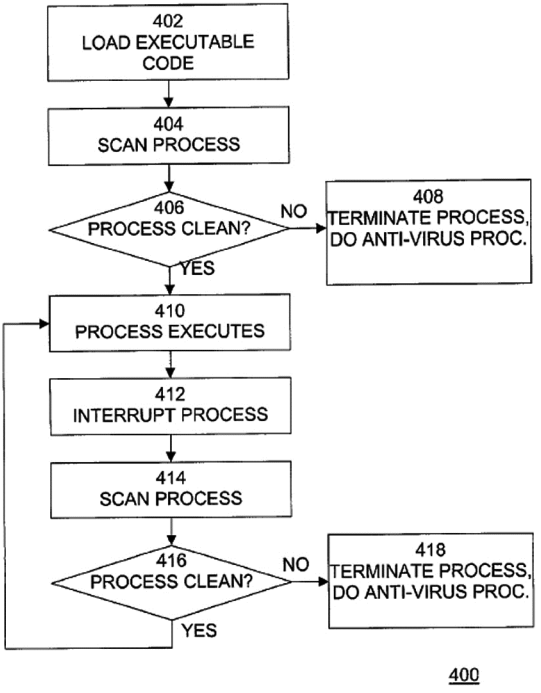
1.2 pav. Patento schema (On-access scan of… 2010)
Fig. 1.2. Patent scheme (On-access scan of… 2010)



1.3 pav. Patento schema (Method and system for preemptive… 2010)
Fig. 1.3. Patent scheme (Method and system for preemptive… 2010)

Pirmiausia surenkama failų prieigos informacija. Antru žingsniu generuojama laiko įkainių statistika, reikalinga failų prieigai. Kitu žingsniu analizuojamas dažnis, kuriuo buvo gauta prieiga prie failo. Po šių žingsnių generuojamas failų prieigos įkainių žymėjimas, ir jie gali būti iš anksto nuskaitomi taip sumažinant skenavimo laiką, reikalingą skenavimo varikliams, reaguojantiems į prieigą prie resursų. Visa patento schema vaizduojama 1.3 paveiksle.

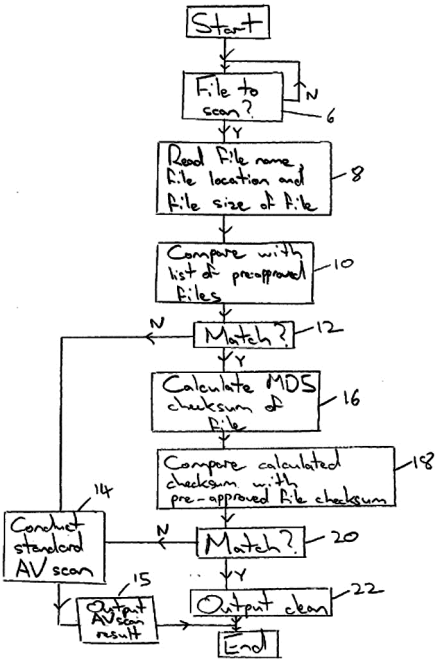
JAV patente (Method and system for detecting… 2003) pristatoma technologija, gebanti aptikti kenkėjišką programinę įrangą supakuotouse ar emuliuojamuose failuose. Metodas pertraukia vykdymo procesą, nuskaito proceso atmintį, ieškodamas kenkėjiško kodo, ir jį leidžia arba nutraukia, priklausomai nuo to, ar rastas kenkėjiškas kodas.



1.4 pav. Blokinė patento diagrama (Method and system for detecting… 2003)
Fig. 1.4. Patent scheme (Method and system for detecting… 2003)

Tokie procesai gali būti susieti su programomis ir įkraunami iš supakuoto ar užšifruoto failo. Failas, kurio nereikia iššifruoti, išpakuoti, gali būti užkrėstas. Išradimo blokinė diagrama atvaizduojama 1.4 pav.

2005 m. užregistruotas JAV patentas „Pre-approval of computer files during a malware detection“. Ši kenkėjiško kodo aptikimo sistema paremta potencialiai užkrėsto failo kontrolinių sumų palyginimu su sveiku failu.



1.5 pav. Patento schema (Pre-approval of computer files… 2005)
Fig. 1.5. Patent scheme (Pre-approval of computer files… 2005)

Jei šios sumos sutampa – potencialiai užkrėstas failas laikomas saugiu. Išradėjai teigia, kad identifikuojant kai kuriuos gerai žinomus failus kaip visiškai švarius, ypač tuos, kurių struktūra gana sudėtinga, gali labai pagerinti antivirusinės programinės įrangos našumą. Atributai, apskaičiuojami iš tokių failų, gelbsti užtikrinti, kad kandidatinis failas nebuvo pakeistas (pvz., operacinės sistemos failas ir t. t.). Principinė šio išradimo schema pateikta 1.5 paveiksle.

1.5. Realiojo laiko skenavimo sistemų failų išdėstymo sistemos stebėjimo metodų analizė

Antivirusinė programa – vienas iš sistemos gynybos mechanizmo komponentų. Ji apsaugo sistemą ir vartotojo duomenis. Antivirusinių programų kūrėjai taiko du pagrindinius peržiūros metodus – pagal pareikalavimą ir realiuoju laiku (pa-

gal prieigą) (Hayes 2010). Peržiūros pagal pareikalavimą atveju vartotojai savo noru aktyvuoja virusų peržiūros programą kiekvieną kartą, kai jie nori patikrinti, ar kompiuteryje nėra virusų.

Realiojo laiko peržiūros atveju apsauga nuo virusų aktyvuojama automatiškai, kai užfiksuojami pokyčiai failų išdėstymo sistemoje (FS) ir (arba) kompiuterio atmintyje.

Realiojo laiko FS stebėjimo sistema dažniausiai analizuoja failus kiekvieną kartą, kai programa kviečia funkciją atidaryti ar uždaryti failą. Atliekant jo atidarymo operaciją, realiojo laiko FS stebėjimo skaitytuvas analizuoja failo turinį ir ieško virusų infekcijos požymių. Tinklo pagrindu veikiančiam virusui (pvz., kirminui) gali niekada neprireikti atidaryti failo, kad užkrėstų sistemą. Vietoje to jis gali patekti į sistemą per vietinį tinklą, sukurti failą, įrašyti į jį duomenis ir tada uždaryti. Skaitytuvas, kuris tikrintų tik failų atidarymą, nesugebėtų užfiksuoti viruso tol, kol nebūtų atidarytas užkrėstas failas (Hayes 2010). Taigi antivirusinėje programoje naudojama realiojo laiko failų stebėjimo sistema gali nulemti kompiuterinės sistemos apsaugą. Šios dalies tikslas – išnagrinėti realiojo laiko FS stebėjimo metodus ir aprašyti jų veikimą.

Failų išdėstymo sistemos stebėjimo būdai. Pasak Szor (2005), realiojo laiko stebėjimo sistema pokyčius FS gali nustatyti šiais pagrindiniais būdais:

1. Veikti kaip programa ir perimti FS API sąsajos funkcijas.
2. Veikti kaip įrenginių/FS filtravimo tvarkyklė, kuri prisitvirtina prie FS (FAT, NTFS ir kt.).

Galima išskirti ir dar vieną būdą – sukurti failų išdėstymo sistemą, skirtą antivirusinei programai.

Galima išskirti dar du failų išdėstymo sistemų stebėjimo metodus – failų kokybės užtikrinimą ir failo naudojimo dažnumo stebėjimą. Stebint failo naudojimo dažnumą, skaičiuojami per trumpą laiką atlikti veiksmai su failu – skaitymas, rašymas į failą, jo atidarymas, uždarymas. Pastarieji du metodai taikomi rečiausiai. Naudojimo dažnumo stebėjimo metodas gali būti neefektyvus stebint FS realiuoju laiku, nes užkrėstas failas gali likti ilgai nepastebėtas, jei bus naudojamas retai.

API sąsajos funkcijų perėmimas. Norint atlikti antivirusinės sistemos darbo optimizavimą, būtina panagrinėti *Windows* API funkcijų stebėjimą. Eksperimentinės sistemos aprašyme bus naudojami gauti stebėjimų rezultatai. API – tai funkcijų grupės, skirtos bendrauti su *Windows* operacine sistema, naudotis jos galimybėmis, ją stebėti. Aptariant eksperimentinę sistemą, reikia plačiau panagrinėti ir taikomus įvairius *Windows* API funkcijų stebėjimo būdus, jų naudojimo ypatumus, galinčias kilti problemas ir paties stebėjimo įtraukimą į veiklą. API funkcijų stebėjimas apima ne tik virusų aptikimui, bet ir tokiam atvejui, kaip derinimui, išsiaiškinti skirtą menkai dokumentuotą API, siekiant praplėsti API funkcionalumą.

Kai kurie virusai kenkia failų sistemai: daugybės failų trynimas, sisteminių failų pakeitimas ir dar keletas kitų būdų. Stebint failų sistemos tvarkymo (I/O) API funkcijas galima labai paprastai stebėti, kas norima daryti, ir įvertinti pasekmes.

Pagrindiniai API stebėjimo būdai:

- naudojant Set*Windows*HookEx() funkciją;
- naudojant Proxy biblioteką (dll);
- pakeičiant IAT;
- pakeičiant programos kodą;
- atliekant failinės sistemos stebėjimą.

SetWindowsHookEx

Pradedama nuo jos, nes API perėmimas su ja yra vienas paprasčiausių ir lengviausiai įgyvendinamas. Pagal aprašymą ji skirta funkcijai įterpti į sistemos funkcijų grandinę. Taip galima stebėti tam tikrus įvykius sistemoje, vieną procesą arba visą sistemą.

Galimi stebėti šiuos įvykius:

- klaviatūros veiksmus;
- pelės veiksmus,
- dialogų veiksmus,
- ir t. t.

Registrai

Norint perimti procesą, esantį USER32.DLL bibliotekoje, tereikia nurodyti savo biblioteką šioje registro šakoje:
HKEY_LOCAL_MACHINE\Software\Microsoft\WindowsNT\CurrentVersion\Windows\AppInit_DLLs

Tai gali būti viena arba kelios bibliotekos. Jos bus paleistos *Windows* programų, naudojančių USER32.DLL biblioteką. Tai nėra vienintelis keblumas taikant šią techniką. Kitas keblumas tas, kad norimas rezultatas bus gautas tik perkrovus sistemą, nes bibliotekos įkraunamos startuojant *Windows* operacinei sistemai. Pagal MSDN šis būdas galioja *Microsoft Windows* NT 4.0, *Microsoft Windows* 2000 Standard Edition ir *Microsoft Windows* XP. Taikant šį metodą reikia atsižvelgti į tai, kad nurodytos bibliotekos bus „prikabintos“ prie visų paleidžiamų programų (naudojančių USER32.DLL), nors tai reikės padaryti tik vienai ar kelioms programoms.

Proxy DLL

Tai taip pat viena paprastesnių API perėmimo funkcijų. Tarkime, norima įgyvendinti antivirusinę programą, kuri skenuoja elektroninį paštą. Tam reikia perimti *Winsock* bibliotekos turimas funkcijas, kurios leidžia bendrauti su serveriu

ir klientu. Tai įmanoma turint įgaliotojo serverio biblioteką, kurioje įgyvendintos visos *Winsock* bibliotekos funkcijos. Žinoma, jos turi buti su identiškais vardais ir identiškomis kvietimo deklaracijomis. Taip pat ši biblioteka turi buti įdėta ten, kur yra elektroninio pašto programa. Taigi kai programa prisijungs wsock32.dll, bus paimtos bibliotekos ir naudojamos funkcijos. O perrašytos funkcijos, atlikusios savo patikrinimus ir skenavimus, turi kreiptis į pagrindines *Winsock* bibliotekos funkcijas. Bet paprastumas kainuoja. Norint pakeisti biblioteką, kuri turi 100 funkcijų, teks visas jas aprašyti. Tai sudėtinga ne tik dėl to, kad tai daug laiko reikalaujantis kruopštus darbas, bet ir dėl to, kad labai dažnai nebūna laisvai prieinamų funkcijų deklaracijų.

IAT modifikavimas

IAT (*Import Address Table*) modifikavimas pagristas principu, kad kiekviena biblioteka (DLL) ar programa yra PE (*Portable Executable*) failo formato. Tai reiškia, kad kiekvienas iš jų sudarytas iš tam tikrų sekcijų. Pavyzdžiui, .text sekcijoje yra sukompiliuotas programos kodas, o .src programa naudoja resursus (dialogus, piktogramas, paveiksliukus). Mus domina .idata sekcija. Joje yra lentelė su naudojamomis funkcijomis ir jų adresais. Tad belieka, paleidus biblioteką ar programą, pakeisti norimos funkcijos adresą į mums palankų (funkcijos adresą).

Programos kodo pakeitimas

Tai ganėtinai sudėtingas API funkcijų perėmimo būdas. Jo esmė tokia: norimos perimti funkcijos pradžioje įrašoma direktyva JMP. Su ja peršokame į norimą funkciją. Kadangi naudojama JMP, į funkciją įterpiami tik keli baitai papildomos medžiagos.

Failų sistemų stebėjimas

Šiuolaikinės antivirusinės sistemos naudoja daugybę failų stebėjimo būdų. Vieni jų aptinka procesus dar prieš juos naudojant, kiti tik susirenka jau atliktus veiksmus.

Pats populiariausias būdas: sistemos tvarkyklė, kuri prijungiama prie pačios failų sistemos ir gali stebėti visą jos aktyvumą.

Kitas būdas – failų kokybės užtikrinimas, kai renkama informacija apie tai, kaip keitėsi faile esantys duomenys.

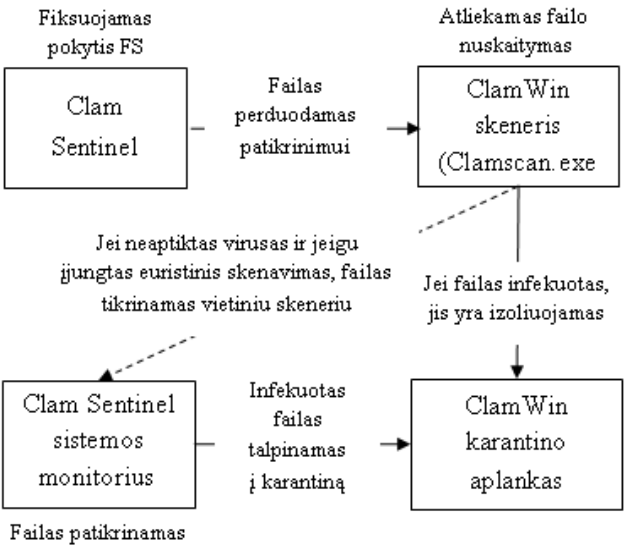
Dar vienas metodas – tai failo naudojimo dažnumo stebėjimas. Kai failas per trumpą laiką yra daug kartų atidaromas, uždaromas, skaitomas ir rašomas, galima jau įtarti kenksmingo kodo veikimą.

API sąsajos perėmimo technikos realizacijos antivirusinėse programose

Šiuo metu populiari atvirojo kodo antivirusinė programa ClamWin, skirta *Windows* OS, neturi galimybės nuskaityti FS realiuoju laiku – šią galimybę suteikia papildomai kompiuterinėje sistemoje įdiegiama realiojo laiko FS stebėjimo programa Clam Sentinel (2009). Failai tikrinami su ClamWin peržiūros varikliu (*Clamscan.exe* failu), naudojant virusų parašų duomenų bazes. Clam Sentinel turi ir savo sistemos stebėtoją, kuris leidžia euristiniu būdu aptikti virusą, kurio parašo nėra ClamWin parašų duomenų bazėje. Supaprastinta Clam Sentinel failų tikrinimo schema pateikta 1.6 pav.

Clam Sentinel naudoja DirMon modulį, kuris informuoja apie aplankų pokyčius – failo/aplanko pasikeitimus (sukūrimo, ištrynimo, pakeitimo, pervadinimo veiksmus). DirMon perima *kernel32.dll* bibliotekoje aprašytą ReadDirectoryChangesW API funkciją. Norint stebėti pokyčius aplanke, sistema turi būti jį atidariusi (*CreateFile()* operacija) (Clam Sentinel 2009). Šios programos gamintojai siūlo Clam Sentinel sistemos monitoriaus aptiktus užkrėstus failus patikrinti ir su kitomis antivirusinėmis programomis (Scroggins 2009).

API sąsajos perėmimo technikas naudoja ir kita atvirojo kodo programa, suteikianti galimybę nuskaityti failus realiuoju laiku – WinPooch, bet ji šiuo metu nebeplėtojama.



1.6 pav. Supaprastina Clam Sentinel failo tikrinimo schema
Fig. 1.6. Simplifies Clam Sentinel file verification scheme

Failų išdėstymo sistemos filtravimo tvarkyklės

Windows tvarkyklėms kurti naudojamas *Microsoft* kompanijos rinkinys – WDK („Windows Driver Kit“). Naujausias rinkinys (WDK 7.1.0) leidžia kurti tvarkykles *Windows 7*, *Windows Vista*, *Windows XP*, *Windows Server 2008 R2*, *Windows Server 2008* ir *Windows Server 2003* operacinėms sistemoms (Windows Driver Kit Version 7.1.0. 2010).

Pradinis kiekvienos tvarkyklės taškas yra funkcija, kuri užregistruoja tvarkyklę filtrų tvarkytuvėje ir inicijuoja visas jos globalias duomenų struktūras. Standartiškai ši funkcija vadinama *DriverEntry*, tačiau gali būti pavadinta ir kitaip.

Jei tvarkyklė sėkmingai užregistruota ir inicijuotos jos duomenų struktūros, grąžinama sėkmingai įvykdytos funkcijos reikšmė *TRUE* (*STATUS_SUCCESS)*, priešingu atveju – atitinkamas klaidos statusas.

Filtrų tvarkytuvė yra *Microsoft* sukurta FS filtro tvarkyklė, kuri skirta trečiųjų šalių filtrų tvarkyklių kūrimui palengvinti (File System Filter Drivers 2012). Kai tvarkyklė registruojama filtrų tvarkytuvėje, nurodoma, kokias operacijas su failais tvarkyklė nori valdyti. Aktualios šios operacijos – failų sukūrimas, atidarymas, įrašymas į juos ir jų uždarymas. Už operacijas su failais atsakingos IRP užklausos. *Microsoft Windows* operacinės sistemos bendrauja su tvarkyklėmis siųsdamos Į/I (įvedimo/išvedimo) užklausų paketus (angl. *I/O Request Packet*, IRP). Šie paketai perduodami tvarkyklių dėkle žemyn nuo vienos tvarkyklės kitai – nuo aukščiausio lygio tvarkyklių iki žemiausio (Opferman 2005). Duomenų struktūra, kuri inkapsuliuoja Į/I užklausų paketus, ne tik aprašo Į/I užklausas, bet ir laiko informaciją apie užklausų, perduodamų tvarkyklėms, kurios jas apdoroja, būsenas (Handling IRPs: What Every Driver Writer Needs to Know 2006). Ši struktūra leidžia tvarkyklėms keistis informacija tarpusavyje (Opferman 2005).

Kai naujas failas ar katalogas yra sukuriamas arba kai egzistuojantis failas, įrenginys, katalogas yra atidaromas, Į/I tvarkytuvė siunčia užklausą IRP_MJ_CREATE. Įrašant failą, siunčiama užklausa IRP_MJ_WRITE, skaitant failą – IRP_MJ_READ. Užklausos IRP_MJ_CLEANUP gavimas parodo, kad failo objekto valdymas yra baigtas. *DriverEntry* funkcijoje tvarkyklės objektui priskiriamos visos IRP užklausos, kurias tvarkyklė turės apdoroti.

Norint sustabdyti tvarkyklės veiklą, joje turi būti aprašyta funkcija, kuri išregistruotų tvarkyklę ir atlaisvintų visas jos globalias struktūras. Šios funkcijos sintaksė aprašyta Unload routine (2012).

Techniškai funkcijos iškvietimas gali būti praleistas, tačiau užregistravus tvarkyklę sistema neleistų atlikti išregistravimo (Opferman 2005). Patikrinti, ar tvarkyklė sėkmingai paleista, ar jos veikla sėkmingai sustabdyta, galima pasinaudojus OSR kompanijos programa *DeviceTree*. Ji parodo sistemoje aktyvius įtaisus ir tvarkykles (DeviceTree 2011). Tvarkyklėse gali pasitaikyti klaidų, kurios gali pažeisti visos operacinės sistemos integralumą. Prieš naudojant tvarkykles, reikėtų įsitikinti, kad jos nesukels problemų (Opferman 2005).

Windows Vista ir vėlesnių 64-bitų *Windows* versijų branduolio-veiksenos kodo pasirašymo politika reikalauja, kad visas branduolio-veiksenos kodas būtų pasirašytas skaitmeniniu parašu. Be to, tam tikros *Windows Vista* ar vėlesnių *Windows* 32 bitų versijų konfigūracijos taip pat reikalauja, kad branduolio veiksenos tvarkyklės turėtų skaitmeninį parašą. Siekiant užtikrinti *Microsoft Windows* platformos saugumą ir stabilumą, minėtos *Windows* versijos pasikliauja skaitmeniniais parašais. Jie leidžia administratoriams ar galutiniams vartotojams, kurie diegia *Windows* sistema paremtą programinę įrangą, sužinoti, ar šią programinę įrangą tiekia legalus leidėjas (Kernel-Mode Code Signing… 2012). Yra būdų, kaip įdiegti nepasirašytą tvarkyklę (Additional Information for Windows Vista… 2010), tačiau jau pats vartotojas sprendžia, ar nepaisyti *Windows* saugumo reikalavimų ir įdiegti skaitmeniniu parašu nepatvirtintą tvarkyklę.

Naujos failų išdėstymo sistemos antivirusinėms programoms

Naujiems FS panaudoti reikalinga virtuali failų išdėstymo sistema (VFS). VFS – sąsaja tarp operacinės sistemos branduolio ir realios failų išdėstymo sistemos (Virtual file system 2012). VFS tikslas – leisti kliento programoms pasiekti įvairaus tipo failų išdėstymo sistemas vienodu būdu. Virtuali failų išdėstymo sistema kviečia žemesnio lygio FS užuot vykdžiusi operacijas pagrindiniame atminties įtaise, pvz., diske ar NFS (Miretskiy *et al.* 2004).

VFS naudojimas populiarus *Unix* ir *Mac* operacinėse sistemose, tačiau gali būti pritaikytas ir *Windows* OS naudojant apvalkalo vardų plėtinius (angl. *Shell namespace extensions*). Tačiau šis būdas turi didelį trūkumą – žemiausio lygio FS nesuteikiama prieiga prie *Windows* API, todėl kai kurios programos gali nepasiekti naujų failų išdėstymo sistemų (Virtual file system 2012). Dėl šios priežasties VFS netinka realiojo laiko FS stebėjimo sistemoms, veikiančioms *Windows* aplinkoje.

Galima išskirti du specifinių failų išdėstymo sistemų tipus:

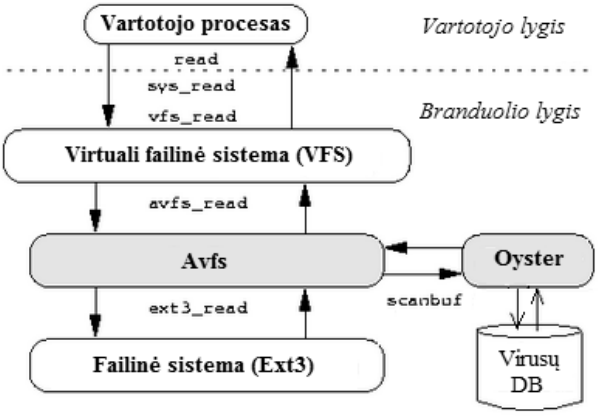
1. Sudurtinė failų išdėstymo sistema.
2. Naujos failų išdėstymo sistemos vartotojo lygmeniu sukurtos naudojant FUSE (Filesystem in Userspace) techniką.

Sudurtinė failų išdėstymo sistema

Sudurtinė FS leidžia esamoms failų išdėstymo sistemos suteikti naują funkcionalumą. Sudurtinė FS iškviečiama virtualios failų išdėstymo sistemos (VFS) kaip ir kitos failų išdėstymo sistemos. Prieš kviesdama kitą FS, sudurtinė FS gali modifikuoti operaciją ar jos kintamuosius. Kviečiama FS gali būti įvairių tipų – Ext2/3, NFS arba netgi kita dėklinė FS.

Miretskiy *et al.* (2004) sukūrė sudurtinę FS *Avfs*, kuri saugo kompiuterinę sistemą nuo virusų. 1.7 pav. rodoma, kaip veikia *Avfs*. Kai *Avfs* prijungiama prie naudojamos FS, ji sukuria tiltą tarp VFS ir naudojamos FS. VFS kviečia įvairias

Avfs operacijas, o tada *Avfs* kviečia atitinkamas failų išdėstymo sistemos, prie kurios ji yra prisitvirtinusi, operacijas. Šių operacijų metu *Avfs* atlieka virusų paiešką. Pavyzdžiui, VFS skaitymo operacija vfs_read() pakeičiama į avfs_read() *Avfs* sluoksnyje. Tada žemesnio sluoksnio skaitymo metodas (ext3_read()) yra iškviečiamas ir gautus duomenis peržiūri *Oyster* programa. *Oyster* – tai Miretskiy *et al.* (2004) realizuotas virusų peržiūrėjimo variklis, integruotas *Linux* branduolyje.



1.7 pav. *Avfs* infrastruktūra (Miretskiy *et al.* 2004)
Fig. 1.7. *Avfs* infrastructure (Miretskiy *et al.* 2004)

Avfs tikrina failus tada, kai programa bando nuskaityti duomenis iš failo ar įrašyti į jį informaciją (Miretskiy *et al.* 2004). Tai vienas iš šios sistemos pranašumų, nes nereikia laukti, kol vartotojas atidarys ar uždarys failą, norėdamas patikrinti jame esančius duomenis . Kenksmingas programinis kodas net nebus įrašytas į failą. Dar viena teigiama savybė ta, kad dėklinė failų išdėstymo sistema, kurios pagrindu veikia Avfs, yra perkeliama – gali veikti skirtingose aplinkose, taip pat gali dirbti ir su įvairiomis FS - Ext2/3, NFS arba kitomis dėklinėmis failų išdėstymo sistemomis. Galima išskirti ir dėklinės sistemos trūkumą – ilgesnė failo nuskaitymo/įrašymo grandinė, nes užklausa perskaityti failo duomenis ar įrašyti į jį informaciją keliauja į naudojamą failų išdėstymo sistemą ne tiesiogiai, bet per stekinę failų išdėstymo sistemą.

FUSE technikos naudojimas

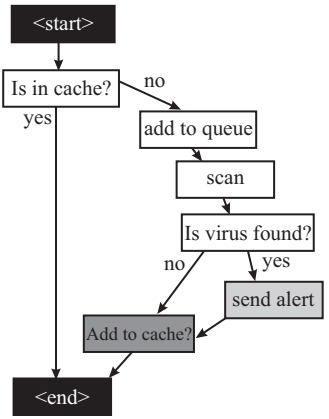
FUSE mechanizmas leidžia vartotojo lygio kodą įterpti į virtualią failų išdėstymo sistemą *Linux*, *NetBSD*, *FreeBSD*, *OpenSolaris* ir *Mac* OS X operacinėse sistemose (Virtual file system 2012).

K. Burghardt yra sukūręs ClamFS – vartotojo lygio failų išdėstymo sistemą, skirtą *Linux* OS, kuri peržiūri failus realiuoju laiku, naudodama antivirusinės programos *ClamAV* skaitytuvą. ClamFS veikia FUSE sistemos pagrindu.

ClamFS programa tikrina failo turinį, kai failas yra atidaromas. Tai yra vienas trūkumų, nes netikrinamos kitos operacijos su failu. Tačiau dėl to programa mažiau apsunkina kompiuterio darbą. Dar vienas programos pranašumų tas, kad ji kaupia patikrintų failų rezultatus ir netikrina to paties failo antrą kartą, dėl to sutaupoma laiko, tačiau čia galima įžvelgti ir trūkumą, nes failą virusas gali užkrėsti ne tik prieš tikrinant, bet ir po to.

1.8 pav. pavaizduota supaprastinta ClamFS failų tikrinimo schema. Iš pradžių pažiūrima, ar failas nebuvo tikrintas anksčiau. Jei failas jau buvo patikrintas, jis iš naujo nėra nuskaitomas. Priešingu atveju failo skenavimas pridedamas į eilę. Atėjus failo peržiūros laikui, jis patikrinamas. Jei failas užkrėstas, sistema išsiunčia įspėjimą. Po kiekvienos peržiūros tikrinimo rezultatas išsaugomas laikinojoje atmintyje (angl. *cache)*.

Vienas iš ClamFS trūkumų tas, kad failas tikrinamas tik jį atidarant ir sistemos užkrėtimas virusu gali ilgai likti nepastebėtas.



1.8 pav. ClamFs failų skenavimo schema (Burghardt 2011)
Fig. 1.8. ClamFs file scanning scheme (Burghardt 2011)

Failų išdėstymo sistemos stebėjimo realiuoju laiku metodų palyginimas

API perėmimo technika ir failų išdėstymo sistemos filtravimo tvarkyklės gali būti panaudotos tik *Windows* operacinėse sistemose. Trečias metodas – specialios failų išdėstymo sistemos sukūrimas dažniausiai taikomas *Unix*, *Mac*, *OpenSolaris* operacinėse sistemose veikiančioms realiojo laiko FS stebėjimo sistemoms. *Windows* operacinėje sistemoje šis metodas nėra tinkamas, nes negali suteikti tokio paties funkcionalumo kaip kitose operacinėse sistemose.

1.2 lentelė. Realiojo laiko skenavimo sistemų palyginimas
Table 1.2. Real-time scanning systems comparison

Realiojo laiko FS stebėjimo programa	Stebimos operacijos su failu	Programavimo kalba	Palaikomos operacinės sistemos	Pranašumai	Trūkumai
ClamFS	Atidarymas	C++	*Linux*	1. Programa veikia greitai, jei dažnai atidaromi tie patys failai	1. Failas netikrinamas jį uždarant 2. Patikrintas failas nebetikrinamas antrą kartą
Clam Sentinel	Sukūrimas, keitimas, ištrynimas	Delphi	*Windows* (98/98SE/ ME/2000/ XP/Vista/7)	1. Tikrinant failą galima remtis ne tik virusų parašų duomenų bazėmis, bet ir euristiniu metodu 2. Tinkamas ir senesnėms *Windows* versijoms.	1. Netikrinama failo uždarymo operacija 2. Dažni netikri virusų aptikimai
WinPooch	Sukūrimas, atidarymas, ištrynimas	C	*Windows 2000*, XP, 2003 (tik 32-bitų versijos)	1. Vartotojo informavimas dėl įtartinų programos veiksmų ir leidimo juos vykdyti klausimas.	1. Netikrinama failo uždarymo operacija 2. Nepalaikomos *Windows* 64-bitų versijos 3. Nepalaikomos *Windows* Vista/7 versijos 4. Galimybė skenuoti tik vykdomuosius failus
Avfs	Skaitymas, rašymas į failą	C	*Linux*	1. Tikrinant įrašomus duomenis į failą, vartotojas neturi jo atidaryti ar uždaryti 2. Dėklinė FS yra perkeliama ir dirba su įvairiomis failų išdėstymo sistemomis	1. Ilgesnis failo skaitymas/rašymas

Visi realiojo laiko FS stebėjimo metodai reikalauja gerai išmanyti metodų veikimą, nes neteisingai įgyvendinti metodai gali pažeisti visos operacinės sistemos integralumą.

API sąsajos perėmimas ir naujos failų išdėstymo sistemos naudojamos atvirojo kodo programose, programų kodas pateikiamas viešai, o FS filtravimo tvarkyklės – komerciniuose produktuose. Kadangi komercinė programa *Norton AntiVirus* – viena geriausių antivirusinių sistemų ir efektyviai apsaugo kompiuterines

sistemas, būtų tikslinga sukurti atvirojo kodo realiojo laiko FS stebėjimo sistemą, naudojančią FS filtravimo tvarkykles.

1.6. Realiojo laiko skenavimo sistemos variklių taikymas

Kiekviena antivirusinė programa gali pateikti klaidingą informaciją apie rastą virusą, kai iš tikrųjų viruso tikrinamame objekte nėra. Kaip teigia Trendy (1996), klaidingi įspėjimai gali sukelti daug rūpesčių ir finansinių problemų kompiuterių vartotojams bei organizacijoms, kai antivirusinės programos mėgina aptikti ir pašalinti neegzistuojantį virusą. Programa, kuri dažnai klaidingai įspėja apie pavojų, yra bevertė. Todėl klaidingų įspėjimų analizė turėtų būti išsamios antivirusinės programos analizės dalis.

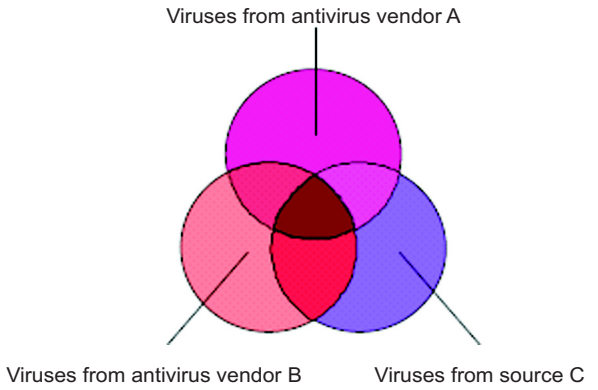
Problema su klaidingų įspėjimų analize yra ta, kad sunku sukurti patikimus rodiklius, matuojančius klaidingų programos įspėjimų jautrumą. Jis negali būti tiksliai įvertintas, nes neįmanoma patikrinti visų dabartinių ir ateities objektų, kurie gali sukelti arba nesukelti netikro pavojaus įspėjimus. Tokio testavimo pagrindas būtų sudarytas iš visų įmanomų dabartinių ir ateities programos kodų, skirtų kompiuterinėms sistemoms, ir, savaime aišku, tokį rinkinį sukurti ir patikrinti neįmanoma.

Vienas galimas būdas klaidingam įspėjimui įvertinti – turėti tūkstančius ar daugiau neužkrėstų failų, kurie būtų skenuojami ir vykdomi, kartu analizuojant antivirusinę programą. Šis metodas nepateikia tikslių rezultatų, tačiau jei bazė gana didelė, klaidingo įspėjimo normos testas galėtų parodyti, kurios programos ar jų nustatymai jautrūs klaidingiems įspėjimams.

Tendencingumo problema. Jei asmuo, analizuojantis antivirusinę programą, ima virusus tik iš vieno antivirusinės programinės įrangos gamintojo ir naudoja juos savo testavimo rinkiniui, labai tikėtina, kad gamintojo antivirusinė programa visus juos ir suras. Tad testavimo rinkinys neparodys teisingų rezultatų, net jei analizė atlikta korektiškai. Tai yra viena didelių testavimo rinkinių problema. Turint omeny, kad nuolat atsiranda naujų virusų, niekas negali būti tikras, kad yra surinkęs visus esamus. Be to, net jei virusų nebebūtų kuriama, niekas negalėtų būti tikras, kad visi esami virusai yra surinkti, nes kai kurie jų gali neegzistuoti šaltiniuose, iš kurių jie buvo imami. Todėl testavimo rinkinys, kuriame yra, kaip teigiama, visi virusai, yra daugiau ar mažiau tendencingas. Šališkumo dydis priklauso nuo to, kiek skirtingų virusų šaltinių analitikas naudojo rengdamas testavimo rinkinį.

1.9 pav. pavaizduota tendencingumo problema. Kiekviena virusų šaltinio grupė turi ne tik užsidengiančias sekas su kitomis grupėmis, bet ir unikalią virusų aibę, kuri neegzistuoja kitose grupėse. Dabar tarkime, kad testavimo rinkinys

neturi virusų iš antivirusinės programos kūrėjo A, tačiau jo produktas vis tiek buvo įtrauktas į analizę. Testavimo rinkinys geresnius rezultatus rodytų antivirusinės programos gamintojui B, o ne A. Testavimo rinkinyje turėtų būti virusų iš abiejų gamintojų. Be to, virusų testavimo rinkinyje taip pat turėtų būti kiti žinomi virusai, kaip kad, pvz., iš gamintojo C (Helenius 2002).



1.9 pav. Tendencingumo problema konstruojant testavimo rinkinį kompiuterinės antivirusinės programos analizei (Helenius 2002)
Fig. 1.9. The problem of bias when constructing a test bed for computer antivirus product analysis (Helenius 2002)

Tendencingumo problema taip pat išskiria į naujus failus iš naujo infekuojamų virusų svarbą ir analizės tik su tam tikros srities virusais problemą. Tačiau galima nesutikti, kad kiti virusai taip pat kelia grėsmę. Vienu atveju galima naudoti didelį testavimo rinkinį, tačiau tokiu atveju reikia jį nuolat atnaujinti naujais virusais iš antivirusinių programų gamintojų. Kaip įprasta, tokia sąlyga ne visuomet įgyvendinama.

Potencialaus pavojaus, keliamo kiekvieno viruso, nustatymas. Būtų idealu atlikti vertinimą tokiu būdu, kad potencialus pavojus, kurį sukeltų kiekvienas virusas, būtų išmatuojamas. Kai kurie virusai plinta labiau nei kiti, todėl labiau tikėtina, kad bus aptikti, negu tie, kurie nesidaugina taip greitai. Deja, atrodo, nėra korektiško mato pavojui nustatyti. Sąrašas gali būti naudojamas kaip matavimo pagrindas, tačiau, kaip jame rašoma, „šis sąrašas neturėtų būti laikomas visų dažniausiai pasitaikančių virusų sąrašu, nes dažnumas nebuvo matuojamas“ (The WildList Organization International 1993–2002). Egzistuoja automatinių programinės įrangos sprendimų, kurie matuoja virusų paplitimą tam tikrose interneto srityse, kaip kad Dr. Solomon Virus Patrol ar Trend Micro World Virus Tracking Center (Trend Micro 2001), tačiau šios priemonės labai ribotos. Pirma-

sis praneša tik apie virusus, išsiunčiamus naujienų grupėms, todėl yra apribotas tik el. pašto virusams ir tam tikro tipo virusų atakoms. Antrasis renka informaciją tik iš savo produktų ir korektiškai neatskiria skirtingų virusų tipų. Todėl reikalingas validus ir dažnai atnaujinamas informacijos šaltinis, matuojantis skirtingų virusų dažnumą.

Be viruso dažnumo, svarbus faktas taip pat yra ir potencialus pavojus, kurį kiekvienas virusas sukelia sėkmingai besidaugindamas. Pvz., jei virusas veikia tik su retai naudojama operacine sistema ar programa, vargu ar nutiks didelių incidentų. Be to, tokios viruso savybės, kaip kad dauginimosi mechanizmas, matomumas ir prisitaikymas, paveikia virusų šansus daugintis. Deja, viruso potencialą efektyviai plisti pamatuoti sunku. Tačiau kruopščiai sukurta klasifikacijos schema, matuojanti dauginimosi potencialą, yra įmanoma ir būtų naudinga.

Virusų parašų didėjimo problema. Aptikus pirmąjį virusą, skirtingų virusų sparčiai daugėjo. Per paskutinius metus virusų skaičius augo beveik eksponentiškai. Vis didėjantis virusų skaičius – akivaizdi problema antivirusinių programų gamintojams, tačiau ji svarbi tiems, kurie nori analizuoti antivirusines programas. Kiekvienas naujas virusas pridės darbo antivirusinės programos recenzentui, nes viruso kopijos turi būti įtrauktos į testinį rinkinį, o naujoms kopijoms patikrinti reikia daugiau produkto įvertinimo laiko. Kadangi egzistuoja tokia daugybė virusų, vienintelis realus būdas patikrinti visus juos – naudoti automatinius virusų dauginimosi mechanizmus arba samdyti daugiau tyrėjų.

Viena problema ta, kad jei norima adekvačiai palyginti analizuotas antivirusines programas, reikia jas imti iš to paties laikotarpio. Tai reiškia, jog antivirusinės programos analitikas turėtų turėti nuolatinę prieigą prie antivirusinių produktų. Jis turėtų tiesiogiai susisiekti su antivirusinių programų kūrėjais ar jų atstovais ir pasinaudoti reguliarių atnaujinimų galimybėmis. Laikinos versijos ir atnaujinimai pasiekiami ir internetu. Specialus produkto siuntimo laiko limitas turėtų būti nenaudojamas, nes įmanoma, kad tai paveiks vertinimo rezultatus. Antivirusinės programos gamintojai gali paviešinti atnaujinimą kaip tik prieš laiko limitą ar netgi pateikti specialią laikinąją versiją. Tokia versija gali gerai veikti aptikimo analizės metu, tačiau jos trūkumas – padidintas jautrumas klaidingiems įspėjimams. Galima reziumuoti, kad reguliari prieiga prie atnaujinimų yra geresnis variantas. Įvertinimo procesas neturėtų paveikti antivirusinės programos tobulinimo proceso.

Virusų atmainų identifikavimo problema. Kartais antivirusinė produkcija gali aptikti tik dalį kopijų. Tai ypač aktualu polimorfinių virusų atveju. Be to, kartais nutinka taip, kad antivirusinis skeneris gali aptikti tik virusą originaliame faile, tačiau ne jo kopijose. Galima nepasisekusio aptikimo priežastis šiuo atveju yra ta, kad antivirusinės programos kūrėjas neįtraukė teisingo parašo ar viruso aptikimo metodo. Todėl svarbu replikuoti virusą į naujus objektus.

Antivirusinė produkcija turėtų būti sukurta tikriems ir veikiantiems virusams, kurie geba rekursyviai replikuotis, aptikti. Visgi tokie virusai ir yra tai, su kuo dažniausia susiduria kompiuterių vartotojai. Todėl Trojos arkliai, linksmos programėlės, nebaigti virusai, pirmosios kartos virusai, neužkrėsti ir sugadinti failai bei kiti nevirusiniai objektai turėtų būti neįtraukiami į testavimo rinkinį. Virusų aptikimo analizė – tai nagrinėjimas, kaip produktai gali aptikti virusus.

Skirtumai tarp virusų atmainų Virusas laikomas besiskiriančiu nuo kitų, jei pagrindinis viruso kodas skiriasi nuo kito bent vienu baitu. Terminas „pagrindinis kodas“ čia vartojamas norint pademonstruoti, kad skirtingos polimorfinio viruso kopijų kartos atstovauja tam pačiai virusui.

Nesvarbu, ar skirtumas yra vykdomame viruso kode, ar viruso duomenų srityje. Viruso duomenų srityje yra duomenų, kurie juda drauge su virusus, tačiau nėra vykdomi. Viruso duomenų pavyzdys – tekstinė žinutė, kurią virusas atvaizduoja ekrane. Duomenų sritį nebūtinai turi naudoti virusas. Pakanka, kad virusas perneša duomenų sritį.

Viena problemų, atskiriant virusų variantus, yra ta, kad virusas dažnai prijungiamas prie pagrindinio kompiuterio, ir viruso kodas turi būti nuo jo atskirtas prieš tiriant tikslią dvejetainę viruso sritį. Gali būti atskiriama rankomis arba, jei pagrindinis kompiuteris yra žinomas, viruso kodą atskirti nesudėtinga. Tačiau polimorfinių virusų atveju negana palyginti binarines viruso formas.

Kita problema ta, kad objektas gali būti užkrėstas keliais skirtingais virusais. Šiuo atveju reikalinga rankinė analizė.

Žinomi virusų skeneriai gali būti naudojami kaip pagalba atskiriant virusų variantus. Jei skeneris aptinka skirtingą viruso variantą mėginio faile, tuomet tai yra puiki priežastis teigti, kad mėginio failuose iš tiesų yra skirtingi virusai. Be to, kai kurie skeneriai gali įspėti, ar pagrindinis kompiuteris užkrėstas daugiau negu vienu virusu. Tačiau skeneriai gali pateikti ir klaidingą informaciją, todėl skirtumai gali būti paaiškinami kitomis priežastimis. Pvz., polimorfinio viruso atveju skeneris gali duoti klaidingą informaciją apie viruso buvimą.

1.7. Ekspertinių sistemų taikymo informacijos saugos sistemose analizė

KPK detektavimas – kritinis informacijos saugos užtikrinimo aspektas. Tradiciniai parašais pagrįsti metodai negali nustatyti „nulinės dienos“ (angl. *zero-day*) atakų, taip pat KPK, kuris taiko įvairius maskavimosi metodus, tokius kaip polimorfizmas, metamorfizmas, slėpimasis ir t. t. Siekiant išvengti parašų metodo ribotumo, siūloma taikyti įvairias anomalijų analize pagrįstas sistemas, kurios savo ruožtu pasižymi dideliu klaidingų pranešimų skaičiumi ir sistemos apmokymo fazės sudėtingumu (Wang 2007). Failų, kurie turi būti išanalizuoti, skai-

čius auga beveik eksponentiškai, nors tik maža dalis failų turės virusų (Le Charlier *et al.* 1995). Perspektyvoje antivirusinių kompanijų ekspertai, kuriantys parašų bazes, tiesiog nespės sukurti juos visiems naujai atsirandantiems KPK (Jacob *et al.* 2008). Ekspertinių sistemų taikymas galėtų būti traktuojamas kaip šitų problemų sprendimas, o būtent leistų pagerinti anomalijomis pagrįstų sistemų patikimumą ir sumažinti analizuojamų failų skaičių, atrenkant pagal vienokius ar kitokius parametrus įtartinus failus.

Ekspertinė sistema – tai programa, kuri specialiai sukurta žmogaus sukauptų ekspertinių žinių taikymui modeliuoti (Ruili 2008). Ekspertinė sistema saugo faktus ir taisykles tokiu pavidalu, kad taikomi kartu jie gali nustatyti naujų faktų. Taip pat ekspertinė sistema turi mokėti paaiškinti, kaip tokios išvados buvo pasiektos (Davis 1992). Ekspertinė sistema naudoja žmonių sukauptų žinių saugyklą siekiant išspręsti problemą, kuriai spręsti tradiciškai reikalingos eksperto žinios. Padengiamos žinios vadinamos žinių sritimi (angl. *domain*). Žinių atvaizdavimas reikalauja sudėtingesnės informacijos atvaizdavimo struktūros, nei yra prieinamos tipinėse duomenų bazėse. Taip atsitinka todėl, kad informacija išgaunama analizuojant neapdorotus duomenis (angl. *raw data*), pridedant papildomos euristinės informacijos (Okafor *et al.* 2007).

Kadangi ekspertinės sistemos yra skirtos spendimų priėmimui pagal ekspertų sudarytus taisyklių rinkinius, tai jų panaudojimas galimas daugelyje informacinių sistemų, tame tarpe antivirusinių sistemų kūrime. Nors pagrindinė antivirusinės sistemos užduotis yra nustatyti ar konkreti byla yra infekuota ar ne, ir daugeliu atveju, šito atsakyti ekspertinė sistema negali, tačiau optimizuojant sistemos darbą ekspertinė sistema gali būti net labai naudinga.

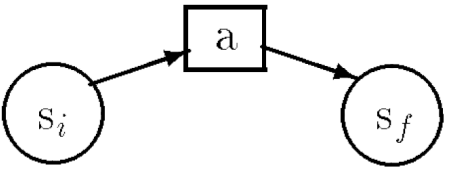
Saugumo srityje šiuo metu ekspertinės sistemos yra labiausiai paplitusios sprendžiant įsiskverbimo detektavimo uždavinius (Ruili 2008). Ekspertinė sistema surenka duomenis monitoringo sistemos priemonėmis. Jei surinkti duomenys yra įtartini, inicijuojamas ekspertinės sistemos taisyklių veikimas ir atliekamas surinktų duomenų automatizuotas ekspertinis vertinimas. Ekspertinės sistemos dažnai pasitelkiamos įsiskverbimui detektuoti dėl pakankamo veikimo tikslumo (mažas klaidingų pranešimų skaičius), didelės greitaveikos, galimybės pagrįsti priimtą sprendimą ir galimybės pildyti sistemos taisyklių bazę, nenaudojant nepageidaujamų tarpusavio priklausomybių. (Ruili 2008).

Ekspertinių sistemų taikymo kovai su KPK istorija yra kur kas trumpesnė ir paplitimo mastas mažesnis. Pirmas straipsnis šia tematika buvo publikuotas 1992 m. (Davis 1992). Jame pasiūlytas CLIPS (*C Language Integrated Production System*) kalbos pritaikymas 12 taisyklių apibrėžimui, kurios ekspertinėje sistemoje gali būti panaudotos kompiuterių virusams detektuoti ir užkrėstos sistemos funkcionalumui atstatyti.

1995 m. straipsnyje (Le Charlier *et al.* 1995) buvo aprašytas ekspertinės sistemos ASAX (angl. *Advanced Security audit trail Analysis on uniX*) pritaikymas

virtualiai kompiuteryje sugeneruotų pranešimų analizei. Generuojami ir legalių programų, ir KPK pranešimai. Ekspertinės sistemos užduotis šiuo atveju – nustatyti pagal tam tikras taisykles, koks programų tipas sugeneravo šiuos pranešimus. Sukurta sistema buvo pavadinta VIDES (angl. *Virus Intrusion Detection Expert System*). Sistemoje apibrėžiamos *a priori* taisyklės, leidžiančios identifikuoti KPK pagal jo veiklos pėdsakus. Sistema buvo orientuota į MS-DOS aplinkoje veikusius virusus ir skirta tik moksliniams tyrimams dėl ypač didelių poreikių procesoriaus resursams. ASAX sistemos taisyklės buvo formuojamos remiantis sukurtu virusų atakų strategijos modeliu, aprašomu RUSSEL kalba, skirta taisyklėms aprašyti ASAX sistemoje.

Atakų scenarijai buvo modeliuojami būsenų pasikeitimo diagramomis (angl. *state transition diagrams*), kuriose mazgai žymi kompiuterinės sistemos būseną, o lankai – programos atliekamus veiksmus, kad būsena būtų pasiekta (1.10 pav.).

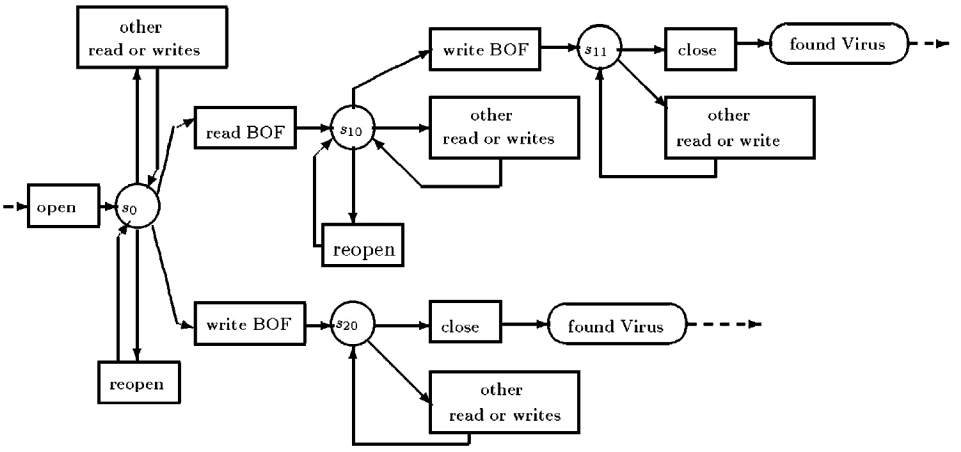


1.10 pav. ASAX sistemos būsenų diagrama Le Charlier *et al.* 1995
Fig. 1.10. Asax system state diagram Le Charlier *et al.* 1995

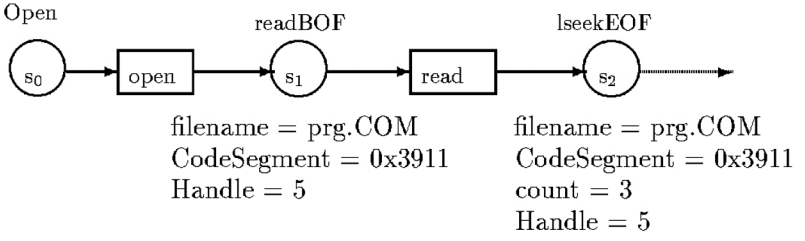
Infekavimo procesas aprašomas kaip veiksmų seka, kuri perveda sistemą iš *švarios* būsenos į finalinę *infekuotą* būseną, kurios metu dalis sistemos failų yra užkrėsta. Taisyklės yra trijų tipų: bendros, tinkamos visiems virusams; specifinės konkrečiam virusui ir kitos. Autoriai teigia, kad tik kelių taisyklių pritaikymas leidžia aptikti daugumą virusų, kurie taiko tipinę infekavimo strategiją. Pavyzdinė būsenų diagrama COM tipo failui infekuoti pateikta 1.11 pav.

Kiekviena būsena diagramoje aprašyta taisykle, kuri ne tik aprašo esamą būseną, bet ir veiksmų seką, kuri prie jos veda (1.12 pav.).

Log įrašai, saugantys informaciją apie legalių programų ir virusų aktyvumą, suformatuoti pagal šabloną, kurį pasiūlė Dorothy Dennings (Denning 1987) įsiskverbimams detektuoti (<Subject, Action, Ob`ject, Exception-Condition, Resource-Usage, Time-Stamp>). Autorių deklaruotas sistemos patikimumas siekia 95 %. Jie taip pat tikina, kad pasiūlytas metodas turi pranašumų, palyginti su parašų detektavimo metodu, nes, skirtingai nuo naujų virusų atsiradimo, naujos virusų infekavimo strategijos pasirodo kur kas rečiau.



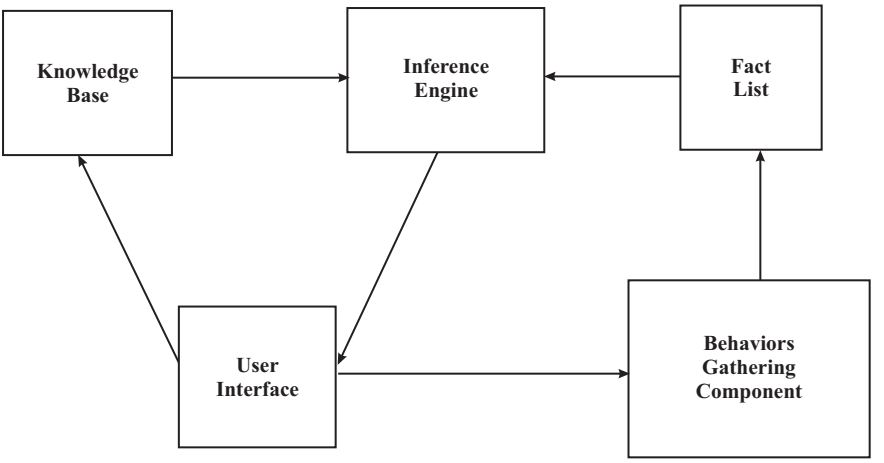
1.11 pav. Būsenų diagrama COM tipo failų infekavimo atveju (Le Charlier *et al.* 1995)
Fig. 1.12. State diagram of COM file type in case of infection (Le Charlier *et al.* 1995)



1.12 pav. Būsenos diagrama, atvaizduojanti būsenos pasiekimo istoriją (Le Charlier *et al.* 1995)
Fig. 1.12. State diagram which visualises state change history (Le Charlier *et al.* 1995)

Dar vienas CLIPS ekspertinės sistemos naudojimas KPK detektuoti buvo pasiūlytas 2008 m. (Ruili 2008). Pasiūlyta sistema integruoja parašais, suprantamus kaip elgesio parašai, paremtą analizę ir anomalijų detektavimo mechanizmus. Remiantis ekspertinėmis žiniomis sukuriama ekspertinės sistemos žinių bazė. Sukuriamas elgesio anomalijų stebėjimo mechanizmas, kuris perduoda perimtą informaciją ekspertinei sistemai vertinti. Sistemos karkasas pateiktas 1.13 pav.

KPK elgesio parašai užrašomi CLIPS kalba. Tipinio rezidentinio viruso elgesio šablonas pateiktas 1.14 pav.



1.13 pav. CLIPS ekspertinės sistemos taikymas KPK detektuoti (Ruili 2008)
Fig. 1.13. CLIPS expert system used for malware detection (Ruili 2008)

```
(defrule check-malicious-process
? allocate -memory <- (allocate-memory
        (srcPrc ?srcprc)(dstPrc ?dstprc&~?srcprc))
? write -memory <- (write-memory
        (srcPrc ?srcprc)(dstPrc ?dstprc&~?srcprc))

?inline-hook <- (inline-hook
        (scrAddr ?srcAddr)(dstAddr ?dstAddr)
        (dstName ?dstName)(modName ?modName))
(test(=(length$ (find-all-instances((?p SYS-PROCESS))
                (eq ?p:name ?srcprc))) 0))
(not (malicious-process (process ?srcprc)))
=>
(printout t ?srcprc " is malicious..." crlf)

(assert (malicious-process (process ?srcprc))))
```

1.14 pav. Rezidentinio viruso elgesio šablono aprašymas CLIPS kalba (Ruili 2008)
Fig. 1.14. Residental virus behavior pattern description language CLIPS (Ruili 2008)

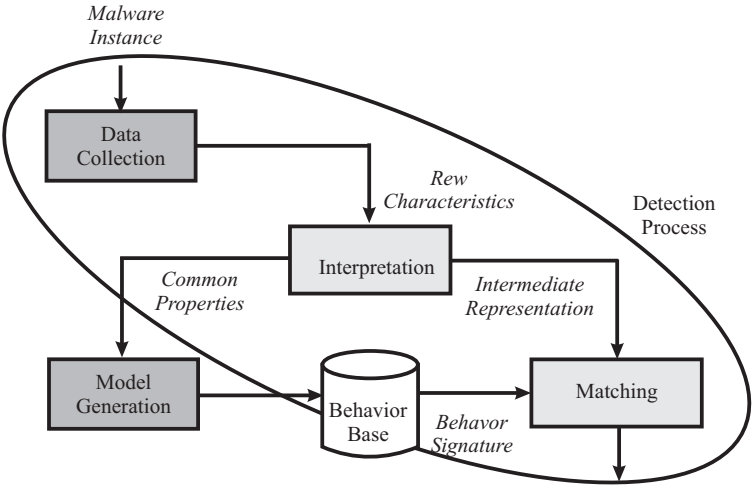
Elgesio surinkimo komponentas (*The Behaviors Gathering Component*) atsakingas už skirtingo tipo informacijos surinkimą iš sistemos, įskaitant KPK elgesio informaciją. Po elgesio informacijos pirminio apdorojimo ji pateikiama kaip faktai. Kiekvieno tipo faktui aprašyti sukuriamas šablonas. Kiekvieno fakto šablono struktūra priklauso nuo elgesio šablono. Sprendimų priėmimo variklis (*Inference Engine*) – tai CLIPS suteikiamas įrankis, kuris automatiškai susieja elgesio šablonus su faktų sąrašu ir nustato, kokios taisyklės turi būti taikomos. Autorių teigimu, sukurtos sistemos detektavimo lygis siekė 100 % (taikant 22 KPK imtį) ir viršijo komercinių produktų, paremtų kodo parašais, patikimumą (Wanga 2009).

2009 m. tas pats autorių kolektyvas pasiūlė elgesio analize paremtą KPK detektavimo sistemą (Wang 2009), kuri apima duomenų gavybos ir ekspertinių sistemų taikomus metodus. Pasiūlytas PrefixSpan* duomenų gavybos algoritmas nustato asociacijas iš KPK elgsenos duomenų bazės ir formuoja KPK elgsenos šablonus (angl. *pattern*), o ekspertinė sistema susieja faktus su taisyklėmis ir priima sprendimą.

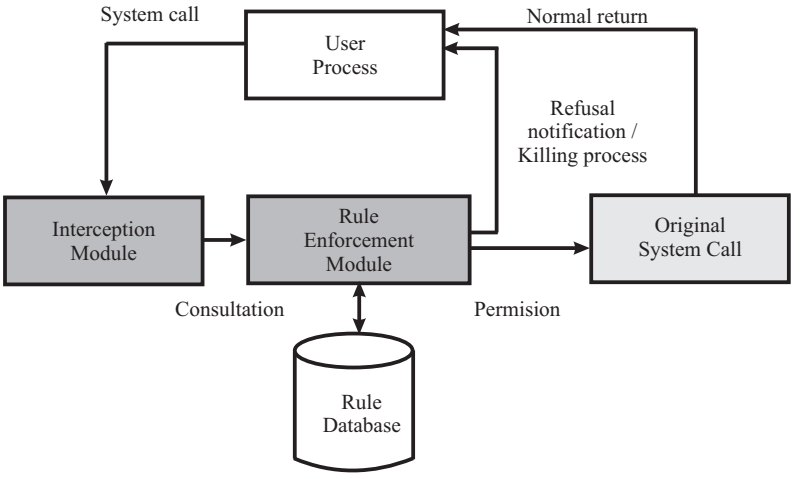
Panašus sprendimas pasiūlytas ir straipsnyje (Jacob *et al.* 2008), kurio autoriai aprašo bendrinį KPK detektavimo modelį, paremtą jo elgesio analize (1.15 pav.).

Taisyklės aprašomos kiekvienam žinomam įtartinam bandymui norint išnaudoti sistemos resursus. Kiekvienas veiksmas, atliekamas stebėjimo programos, dinamiškai susiejamas su atitinkamomis taisyklėmis. Kviečiančios programos tikslas ir sisteminis lygis yra ypač svarbūs veiksniai analizės metu, nes jie dažnai nulemia, ar veiksmas bus priskirtas prie teisėtų, ar prie kenkėjiškų veiksmų (1.16 pav.).

Kiekvienam perimtam kvietimui aktyvuojamos atitinkamos taisyklės. Pagal taisyklę sistema perduoda kontrolę kviestai funkcijai arba siunčia atsisakymo / proceso užmušimo (angl. *kill*) signalą kviečiančiam procesui.



1.15 pav. Elgesio analize paremto detektavimo mechanizmo funkcinė architektūra (Jacob *et al.* 2008)
Fig. 1.15. Conduct analysis based detection mechanism functional architecture (Jacob *et al.* 2008)



1.16 pav. Taisyklių susiejimas su analizuojamais veiksmais (Jacob *et al.* 2008)
Fig. 1.16. Rules compliance with the analyzes of (Jacob *et al.* 2008)

Kaip matoma iš pateiktos ekspertinių sistemų taikymo antivirusinėse sistemose analizės, moksliniai tyrimai telkiami tik į jų taikymą KPK detektuoti, o kitas perspektyvus taikymo aspektas lieka nepaliestas, t. y. nėra tyrimų, aprašančių ekspertinių sistemų taikymą skenuojamo failo srautui minimizuoti, kuris, didėjant operacinių sistemų ir taikomosios programinės įrangos sudėtingumui, tampa vis aktualesnis.

1.8. Informacijos saugos sistemų projektavimo metodai

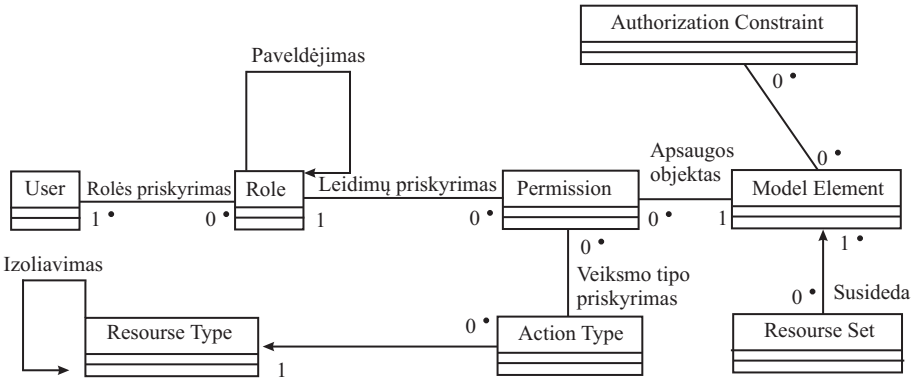
Projektuojant bet kokią sistemą saugumo klausymas yra be galo aktualus. Jau beveik nebeįmanoma rasti sričių, kuriuose kuriant IT sistemą, jos saugumas gali būti pamirštamas. Bendruoju atveju technologija kaip prieigos kontrolė gali būti modeliuojama ir skirstoma į dvi kategorijas: savarankišką ir nesavarankišką.

SecureUML

SecureUML (Lodderstedt *et al.* 2002; Doser 2003) yra UML išplėtimas RBAC specifikavimui ir kitoms prieigos kontrolės taisyklėms, ribojančioms veiksmus apsaugotuose resursuose, naudojant saugumo modeliavimo kalbą. Abstrakti sintaksė ir semantika leidžia ją sujungti su architektūros modeliavimo kalbomis. Ši

metodika yra RBAC kalba, išplėsta autorizacijos apribojimais, kurie išreikšti objektų apribojimo kalba (angl. *Object Constraint Language*, OCL).

1.17 pav. vaizduojamas SecureUML metamodelis UML klasės diagramoje. SecureUML metamodelio tipai pažymėti šviesesne spalva. Šis metamodelis yra UML metamodelio plėtinys. SecureUML turi naujus metamodelio tipus *User (Vartotojas), Role (Vaidmuo)* ir *Permission (Leidimas).* Ryšiai tarp šių tipų apibrėžiami kaip standartinės UML asociacijos. Dar vienas tipas, pristatytas SecureUML, yra *AuthorizationConstraint (AutorizacijosApribojimas)*, naudojantis standartinį UML branduolio tipą *Constraint (Apribojimas)* išreiškiant sąlygas vartotojo apibrėžtų resursų aibės operacijų kvietimams.



1.17 pav. SecureUML metamodelis
Fig. 1.17. SecureUML metamodel

UMLsec

Kitas UML plėtinys saugioms sistemoms kurti yra UMLsec, pristatytas J. Jürjens (2004). Čia pasikartojantys saugumo reikalavimai, kaip kad slaptumas, integralumas, autentiškumas ir t. t., yra specifikacijų elementai. Įvairios UML diagramų rūšys naudojamos galimiems pažeidžiamumams parodyti.

UMLsec plėtinys suteikiamas standartinių UML plėtinių mechanizmų naudojimo, kaip kad stereotipai, žymės ir apribojimai, forma. Stereotipai su žymėmis formuluoja saugumo reikalavimus, o apribojimai suteikia kriterijų, ar reikalavimai tenkinami sistemos architektūros, ar ne.

Pagrindinė UMLsec plėtinio idėja – apibrėžti žymas UML modelio elementams (stereotipams), kurie, kai priskiriami, pridėtų su saugumu susijusią informaciją prie šių modelio elementų (Jürjens 2004). Tokia žyma yra «rbac» stereotipas, kuris bus paaiškintas kituose skyriuose.

Pareigų atskyrimas

RBAC pareigų atskyrimo (angl. *separation of duty*, SOD) apribojimai naudojami interesų taisyklių konfliktams įgyvendinti (Ray *et al.* 2004). Šie apribojimai gali būti dviejų tipų:

- Statiškas SOD (SSD) – neleidžia interesų konfliktų, kurie kyla, kai vartotojas gauna leidimus, susietus su konfliktuojančiais vaidmenimis, t. y. vaidmenimis, kurie negali būti priskirti tam pačiam vartotojui.
- Dinamiškas SOD (DSD) – įveda apribojimą vaidmenims, kurie negali būti aktyvuoti tos pačios vartotojo sesijos metu.

Statiniai ir dinaminiai SOD apribojimai gali būti pridedami prie leidimų ar vartotojų, o ne vaidmenų. Priklausomai nuo objekto, prie kurio pridedami apribojimai, šie vadinami SSD vaidmuo, SSD leidimas arba SSD vartotojo apribojimai.

1.9. Pirmojo skyriaus išvados

Skyriuje aptarta kenksmingo programinio kodo techninė analizė, jo raida ir pagrindiniai KPK tipai. Išanalizuoti rinkoje esančių realiojo laiko skenavimo antivirusinių sistemų variklių veikimo principai bei jų problematika. Susistemintos vartotojo veiksmų operacinėje sistemoje stebėjimo realiuoju laiku metodikos ir galimybės.

Buvo apžvelgti stebėjimo, kenksmingo programinio kodo ir anomalijų aptikimo varikliai mobiliosioms technologijoms, aprašyti teigiami ir neigiami jų aspektai. Pristatytos patentuotos, komercinės ar uždarojo kodo technologijos, siekiant supažindinti su įvairiomis technologijomis ir metodais, taikomais realiojo laiko skenavimui komerciniuose produktuose.

Išanalizuotas ekspertinių sistemų taikymas informacijos saugos sistemų kūrimo srityje. Palygintos skirtingos saugaus projektavimo metodikos, kurios bus naudojamos sistemos architektūrai kurti.

2

Siūlomos realiojo laiko skenavimo sistemos architektūra

Pirmojoje disertacijos dalyje išanalizuota realiojo laiko skenavimo sistemos virusų aptikimo, variklių standartų, patentų, disertacijos metu darytų eksperimentų su API funkcijomis, ekspertinių sistemų taikumo metodais visuma. Remiantis atlikta analize nustatyta, kad neegzituoja nepatentuotu efektyvių realiojo laiko skenavimo antivirusinių sistemų taikančių ekspetinių sistemų principus. Pasiūlytas realiojo laiko skenavimo antiviruso prototipas, kurio projektas aprašomas šioje dalyje.

Sistemos architektūros aprašymas suskirstytas į funkcinių ir nefunkcinių reikalavimų aprašymus, bei sistemos projektą. Projekte pateiktos svarbiausios UML bei UMLSec diagramos ir kitos schemos apibūdinančios sistemos sandarą.

Skyriaus tematika paskelbti trys autoriaus straipsniai (Čenys *et al.* 2009; Radvilavičius *et al.* 2012; Radvilavičius, Talmontienė 2012).

2.1. Funkciniai reikalavimai

Antivirusinės sistemos kūrimo pirminėje stadijoje, kaip ir bet kurios kitos sistemos kūrimo procese, būtina suformuluoti kuo tikslesnius funkcinius reikalavimus. Juos taikant užtikrinamas tinkamas sistemos darbas, pildantis visus iškeltus reikalavimus. Prieš kuriant sistemą buvo iškelti šie funkciniai reikalavimai:

- Sistemos architektūros reikalavimai.
- Resursų naudojimo ribos.
- Realiojo laiko skenavimo mechanizmo palaikymas.
- Operatyviosios atminties skenavimo funkcija.
- Nurodyto katalogo ar disko skenavimo funkcija.

Architektūriniai sistemos reikalavimai

Architektūriniai reikalavimai – vieni svarbiausių funkcinių reikalavimų, nes būtent jais remiantis nustatomos svarbiausios pagrindinių sistemos darbo principų, modulių, jų tipų bei komunikacijos tarp jų. Kuriamą sistemą turėtų sudaryti šie moduliai:

- Pagrindinė parašų duomenų bazė – joje bus laikomi visi KPK parašai. Ši duomenų bazė bus didelės apimties, nes turės parašus nuo pat jų rinkimo pradžios.
- Naujausių parašų duomenų bazė – šioje duomenų bazėje bus saugomi paskutinių kelių dienų ar savaičių parašai. Toks duomenų bazių atskyrimas būtinas norint pasiekti didesnę greitaveiką bei optimizuoti sistemos darbą.
- Antivirusinės sistemos ClamAV programa, kuri dirbs kaip operacinės sistemos tarnyba foniniu režimu. Šio modulio tikslas – sulyginti pateiktą bylą su nurodytų duomenų bazių parašais ir pateikti atsakymą.
- Antivirusinės sistemos programa, kuri dirbs kaip operacinės sistemos servisas foniniu režimu. Ši dalis – viena svarbiausių visoje sistemoje, nes būtent čia bus atliekami visi pagrindiniai veiksmai ir dalijamos užduotys kitiems posistemiams. Šioje dalyje bus realizuotas ir ekspertinės sistemos modulis.
- Operacinės sistemos sekų perėmimo modulis. Jis atsakingas už vartotojo veiksmų failų sistemoje stebėjimą.
- Vidinės duomenų bazės – tai dažniausiai naudojamų bylų ir bylų kontrolinių sumų duomenų bazės.

Resursų naudojimas

Kaip jau minėta pirmoje darbo dalyje, šiam projektui ypač svarbus nedidelis resursų naudojimas. Dėl to buvo iškelti žemi minimalūs reikalavimai kompiuterinei įrangai, kad sistema funkcionuotų tinkamai. Reikalavimai buvo parinkti

atsižvelgiant į mobiliųjų įrenginių (mobilieji telefonai, planšetiniai kompiuteriai ir pan.), šią dieną išleistų į rinką, charakteristikas:

- Procesorius: 800 Mhz.
- Operatyvioji atmintis: 32 MB.
- Kietasis diskas: 40 MB.

Visų kitų kompiuterio resursų prieinamumas neturi turėti jokios reikšmės sistemos darbui.

Realiojo laiko skenavimo mechanizmo palaikymas

Vienas pagrindinių šio darbo tikslų yra, tas, kad kuriama antivirusinė sistema palaikytų realiojo laiko vartotojo veiksmų aptikimo, perėmimo ir kontrolės funkcijas. Sistema turi stebėti visus operacinėje sistemoje veikiančius procesus, fiksuoti jų veiksmus su failų sistema. Užfiksavusi tam tikrą veiksmą, sistema jo duomenis turi perduoti patikrinti, o gavusi atsakymą – veiksmą blokuoti arba leisti vykdyti.

Operatyviosios atminties skenavimo funkcija

Sistemai taip pat keliamas reikalavimas – patikrinti kompiuterio operatyvinę atmintį. Ši funkcija reikalinga, kad ir pati sistema, ir vartotojas galėtų įsitikinti, ar KPK šiuo metu nėra apkrėtęs operatyviosios atminties.

Nurodyto katalogo ar disko skenavimo funkcija

Kaip ir daugelis antivirusinių programų, taip pat ir mūsų kuriama, turi palaikyti nurodyto katalogo ar disko skenavimą. Ši funkcija būtina norint turėti galimybę aptikti KPK, kuris dar nėra paleistas, kopijuojamas ar modifikuojamas realiuoju laiku, bet kažkada buvo įkeltas į elektroninę laikmeną.

2.2. Nefunkciniai reikalavimai

Tam, kad būtų sklandžiai vykdomi funkciniai reikalavimai, būtina užtikrinti ir tinkamą nefunkcinių reikalavimų aibę, kurie yra ne mažiau svarbūs nei funkciniai. Nefunkciniais reikalavimais užtikrinamas sklandus darbas, greitaveika, vartotojo pojūčiai dirbant su sistema ir pan.

Ikiprojektinėje stadijoje buvo atrinkti šie nefunkciniai reikalavimai:

- Greitaveika.
- Patogumas vartotojui.
- Saugumas.
- Sistemos darbas neturi paveikti įprasto operacinės sistemos funkcionalumo.
- Sistemos darbas neturi paveikti taikomųjų programų įprasto funkcionalumo.
- Vartotojo veiksmų užlaikymo trukmė.

Greitaveika

Greitaveika – tai vienas pagrindinių reikalavimu kalbant apie antivirusines sistemas. Kadangi antivirusinės sistemoms paskirtis – dirbti foniniu režimu, o pagrindinis funkcinis reikalavimas – tikrinti vartotojų vykdomas operacijas, todėl nuo jo greitaveikos tiesiogiai priklauso laikas, kuriuo užlaikomas kitų sistemų darbas.

Mūsų atveju tinkamas greitaveikos matavimo ir skirtingų sistemų palyginimo rodiklis yra tos pačios operacijos atlikimo laikas, lyginant sistemą su antivirusine programa ir be jos arba su keliomis skirtingomis.

Šiai bandomajai sistemai keliamas reikalavimas išlaikyti greitaveikos lygį, kuris ne daugiau kaip 20 proc. nusileistų šiuo metu geriausioms komercinėms sistemoms.

Patogumas vartotojui

Patogumas vartotojui yra gan svarbus dalykas, nors pati sąvoka yra gana abstrakti. Patogumas daugiausia susijęs su kitų nefunkcinių (o kartais ir funkcinių) reikalavimų įvykdymo visuma. Taip pat vartotojo sąsaja, kuri turi būti paprasta ir intuityvi.

Naudojant tokio tipo sistemas bene svarbiausias vartotojo kriterijus – sistemos „netrukdymas“ darbui, t. y. sistema turi atitikti jau minėtus greitaveikos bei suderinamumo reikalavimus.

Saugumas

Kadangi antivirusinė sistema skirta kompiuterio ir vartotojo apsaugai, jos pačios saugumo klausimas yra labai svarbus. Pažeidus šį reikalavimą atsiranda tikimybė, kad priemonė, atsakinga už saugojimą, gali būti valdoma KPK, dėl to prarandamas pasitikėjimas ja.

Tam, kad sistema būtų kuo saugesnė, būtina taikyti saugumo užtikrinimo metodus nuo pat sistemos projektavimo pradžios. Taip pat būtina pritaikyti saugaus programavimo metodus rašant pačios sistemos programinį kodą.

Sistemos darbas neturi daryti poveikio įprastam operacinės sistemos funkcionalumui.

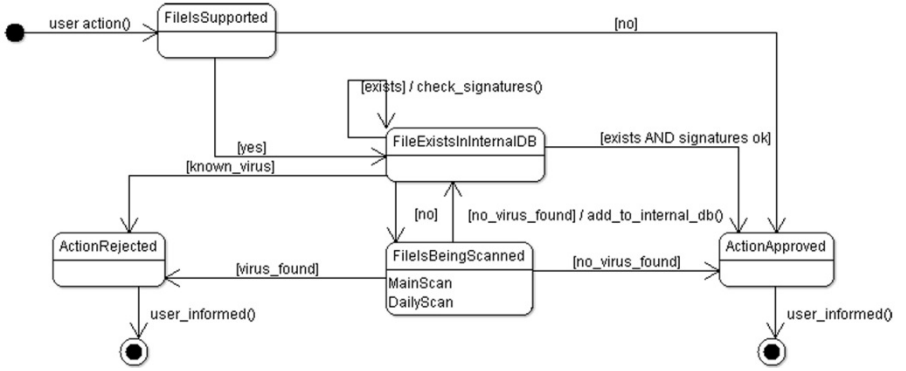
Kaip ir kiekviena antivirusinė sistema, taip ir ši neišvengiamai turi įsiterpti į tam tikras operacinės sistemos funkcijas ir bibliotekas (detalesnis metodų taikymas aprašyta kituose skyriuose). Tai gali turėti ir šalutinį poveikį, todėl šis nefunkcinis reikalavimas yra būtinas, ir turi būti užtikrinta jau nuo projektavimo stadijos, o jo įvykdymas turi būti užtikrintas testuojant sistemą.

2.3. Realiojo laiko skenavimo antivirusinės sistemos projektas

Realiojo laiko skenavimo antivirusinės sistemos būsenų diagrama

Ši būsenų diagrama atvaizduoja svarbiausias kuriamos sistemos veiklos būsenas ir jų kitimą toje pačioje sistemoje. Ji nusako kuriamo prototipo objektų būsenas ir jų pasikeitimus laiku einant. Būsenos diagrama žingsnis po žingsnio parodo sistemos komponentų srautus.

Toliau pateikiama esminių sistemos elementų dalinė paprastoji UML būsenų diagrama. Ji vaizduoja esminių sistemos komponentų būsenas įvairiomis sąlygomis ir skirtingais laiko momentais.



2.1 pav. Sistemos būsenų diagramos realizacija UML kalba
Fig. 2.1. System state diagram in UML

Diagrama prasideda nuo veiksmo, kuri inicijuoja kompiuterio vartotojas (user action()), iš karto po to veiksmas patikrinamas įvairiose duomenų bazėse. Schemoje pavaizduota palaikomųjų bylų duomenų bazė (FileIsSupported), kuri patikrina, ar bylą apskritai liepta tikrinti. Jei tikrinti nebūtina, veiksmas patvirtinamas ir vartotojui leidžiama jį vykdyti.

Kita diagramos šaka yra kur kas sudėtingesnė, nes jei patikrinus palaikomųjų bylų duomenų bazėje (FileIsSupported) paaiškėja, kad byla priskirta prie tikrintinų. Tada veiksmas turi pereiti visą patikros mechanizmą. Šioje diagramoje pavaizduotos dvi galimos funkcijos ir jų sąveika (FileExistsInInternalDB ir FileIsBeingScanned), po kurių vartotojo sukeltas veiksmas patvirtinamas arba atmetamas, t. y. blokuojamas priėjimas prie bylos.

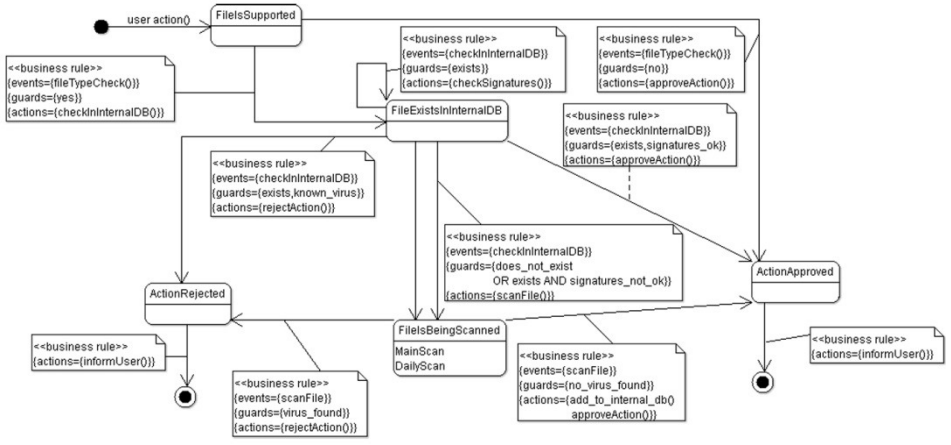
Patekus į vidinės duomenų bazės patikrą (FileExistsInInternalDB), jei byla yra vidinėje duomenų bazėje, patikrinama, kokie buvo *main* ir *daily* duomenų

bazių numeriai. Aptikus kurį nors nesutapimą, rinkmena nuskaitoma su nesutapusia virusų parašų duomenų baze. Tokiu būdu užtikrinamas tik reikiamos apimties patikrinimas. Jei bent vieno iš patikrinimų metu rastas KPK – grąžinama reikšmė, kad failas užkrėstas. Švari rinkmena įrašoma į vidinę duomenų bazę (FileExistsInInternalDB) ir priimamas atitinkamas sprendimas.

Kadangi kuriama sistema yra ypač jautri saugos klausimams, todėl jau projektuojant į tai turi būti atsižvelgta. Tai labai svarbu padaryti pirmose stadijose, kad kiti proceso dalyviai, tokie kaip sistemos programuotojai, turėtų kur kas mažiau laisvės interpretacijai, ir sistemos saugos klausimai neliktų priklausomi vien tik nuo programuotojo gebėjimų šioje srityje.

Buvo pasirinktas UMLSec standartas, kurio galimybės leidžia aprašyti ir IT saugą užtikrinančius stereotipus. Kiekvieną iš perėjimų ankstesnėje diagramoje perbraižius, naudojant naujuosius UMLSec, gaunama kur kas detalesnė schema.

Kaip matoma iš UMLsec diagramos, verslo taisyklė gali būti specifikuota praleidžiant vieną iš žymių (įvykį, veiksmą ar sąlygą) arba naudojant sudėtinę žymę (kelias sąlygas ar kelis veiksmus).



2.2 pav. Sistemos būsenų diagrama realizuota UMLSec standarto pagalba
Fig. 2.2. System state diagram implemented in UMLSec

Kadangi stereotipai leidžia pateikti kur kas daugiau nedviprasmiškos informacijos apie būseną ir jos kaitą, projektas tampa kur kas aiškesnis ir griežtesnis. Tai įpareigoja tolesnius sistemos kūrimo dalyvius laikytis nustatytų taisyklių.

Realiojo laiko skenavimo antivirusinės sistemos kompozicinės struktūros diagrama

Kadangi viena svarbesnių tokio tipo sistemų savybių – sistemos greitaveika, nes vartotojui jos darbas turi likti mažiau pastebimas, labai svarbu užtikrinti kritinių sistemos modulių komunikacijos efektyvumą. Šiuo atveju, renkantis komunikavimo metodą iš modulių, pagrindinis tikslas (*{goal}*) – sistemos veikimas, o jis nusakomas konkrečiais tikslais:

1. Uždelsimas nuskaitant ir tikrinant turi būti minimalus (diagramoje – immediate (T_{min})).
2. Užklausų praradimas yra netoleruotinas, t. y. visos užklausos (*eventual*) 100 % turi būti apdorotos ir grąžinti atsakymai.

Programuojant sistemos komponentus, turi būti nustatytas geriausias galimas būdas, atitinkantis abu šiuos tikslus. Todėl turėtų būti sudaryta galimų alternatyvų aibė (komunikavimo metodai), tada bandymų būdu nustatoma mažiausio uždelsimo alternatyva (1 tikslas). Alternatyvos, kurios lemia užklausų praradimą (<100 %), turi būti atmetamos (2 tikslas).

Toliau aprašytas pasirinkimas gali būti grįstas UMLsec stereotipais <<performance>>, <<call>>, <<send>>, <<guarantee>> bei kitais elementais.

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 nusiuntusiam procesui.

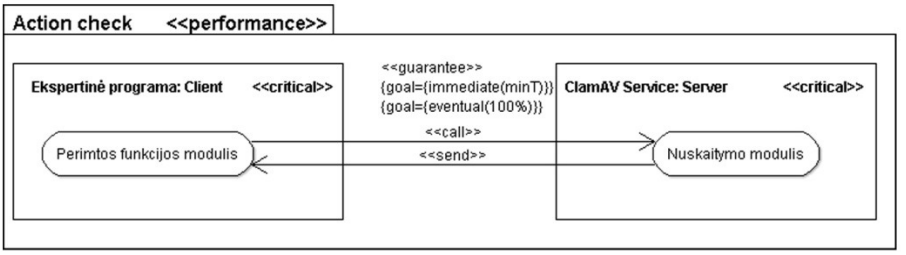
Stereotipų paaiškinimai:

- <<call>> – metodų kvietimas (šiuo atveju nuskaitymo modulio metodų kvietimas);
- <<send>> – signalų siuntimas (šiuo atveju rezultato grąžinimas);
- <<performance>> – sistemos veikimas funkcine prasme;
- <<guarantee>> – funkcinių sistemos reikalavimo užtikrinimas.

Žymėtųjų reikšmių (*tagged values*) paaiškinimai:

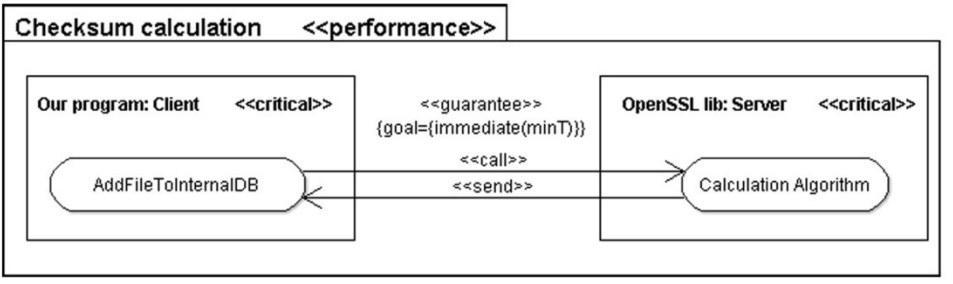
- {goal} – savaime suprantamas tikslas arba tikslas, aprašytas detalesnėms reikšmėms:
  - immediate(t) – galimas uždelsimas (*delay*), čia *t* – laikas;
  - eventual(p) – tikimybė *p*, kad rezultatas grąžintas per laiką *t*.

Toliau pateikiama diagrama.



2.3 pav. Veiksmo patikros digrama
Fig. 2.3. Action composit structure diagram

Grindžiant kontrolinės sumos skaičiavimo algoritmo pasirinkimą, galima naudoti <<performance>> ir <<guarantee>> stereotipus. Čia tikslas taip pat yra minimalios laiko sąnaudos – immediate ($T_{\min}$).

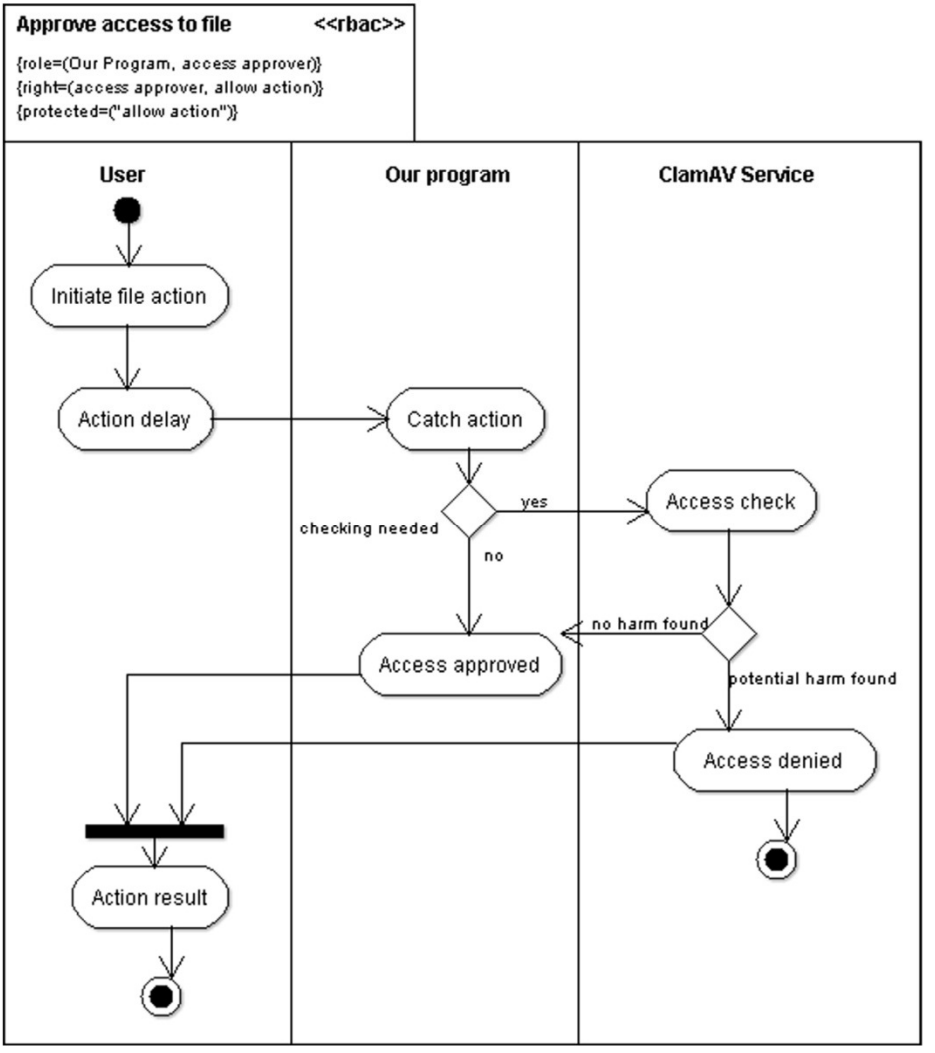


2.4 pav. Maišos funkcijos vykdymo diagrama
Fig. 2.4. Hash function composit structure diagram

Realiojo laiko skenavimo antivirusinės sistemos veiklos diagrama

Veiklos diagrama padeda grafiškai atvaizduoti darbo eigą ir sąsajas tarp skiritngų sistemos komponentų numatant kelis galimus kelius. Veiklos diagrama leidžia žingsnis po žingsnio atkartoti visą sistemoje projektuojamą procesą.

Standartinę UML veiklos diagramą papildėme UMLSec stereotipu RBAC (ang. *role-based access control*). Esminis šios sistemos darbo rezultatas – atsakymas, leidžiantis arba draudžiantis toliau vykdyti vartotojo sukeltą veiksmą. Todėl būtina pasirūpinti, kad šį sprendimą priimtų tik tas sistemos komponentas, kuris už tai atsakingas.



2.5 pav. Sistemos veiklos diagrama
Fig. 2.5. System activity diagram

ClamAV tarnyba jokių specialių teisių neturi – prie apsaugoto ištekliaus prieigą suteikia mūsų sistema (*our program*), o ClamAV tik atlieka patikrinimo funkciją. Apsaugotas išteklius turi būti prieinamas tik jūsų sistemai, todėl jis yra specifikuojamas {protected} žyme.

Schemoje yra dar vienas dalyvis – User, tačiau jis neturi jokių vaidmenų ir kartu betarpiškų teisių į saugomus išteklius/objektus.

Sistemos techninių sprendimų pasirinkimo kriterijai

Toliau pateikiamas perėmimo technologijų ir laikų palyginimas remiantis Hunt, Brubacher (1999) ir sulyginamas su kitais API perėmimo mechanizmais.

2.1 lentelė. Perėmimo technologijų palyginimas
Table 2.1. Interception technique comparison

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

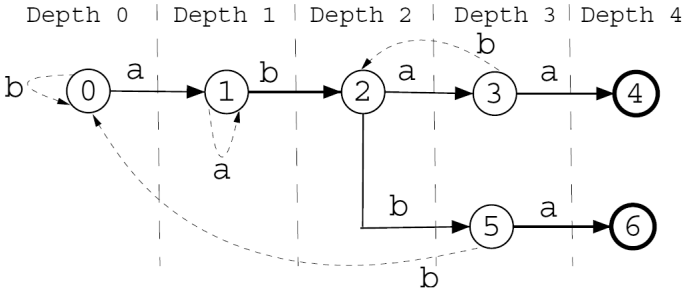
Iš šių testų aiškiai matyti, kaip svarbu pasirinkti ir suprojektuoti prieigą prie resursų korektiškai, siekiant taikyti realiojo laiko skenavimo metodus.

Atsižvelgiant į anksčiau nustatytus kriterijus, buvo nuspręsta naudoti nemokamą Clam AntiVirus (ClamAV) (http://www.clamav.net) skenerį kaip pagrindą realiojo laiko taisyklėmis paremtam antivirusinės programinės įrangos varikliui. ClamAV susideda iš pagrindinės skenerio bibliotekos, taip pat įvairių komandinių eilučių programų (Miretskiy *et al.* 2004), tačiau jame nėra realiojo laiko skenavimo metodo. Taigi buvo pakoreguota ClamAV skenerio biblioteka, skirta naudoti su kurtuoju servisu ir realiojo laiko apsauga.

ClamAV Virus Database 2012 m. spalio mėn. turėjo 1 044 360 parašų. ClamAV virusų apibrėžimų duomenų bazėje yra dviejų tipų virusų šablonai: 1) primityvūs šablonai, suformuoti iš įprastos simbolių sekos, kuri identifikuoja virusą; 2) daugiadaliai šablonai, susidedantys iš daugiau nei vieno primityvaus šablono. Kai visi daugiadalio šablono pašabloniai atitinka viruso šabloną tam tikra tvarka, jis laikomas užkrėstu. ClamAV virusų šablonai taip pat gali turėti pakaitos (*) simbolį. Daugiadalių šablonų ir pakaitos simbolio junginys leidžia ClamAV aptikti polimorfinius virusus. Polimorfiniai virusai sunkiau aptinkami negu nepolimorfiniai, nes kiekvienas atskiras viruso atvejis turi skirtingą pėdsaką iš kitų atskirų atvejų (Miretskiy *et al.* 2004).

ClamAV naudoja Aho-Corasick šablonų atitikimo algoritmo variantą, kuris gerai tinkamas programoms, kuriose didelis kiekis šablonų lyginamas su įvesties tekstu.

Algoritmas veikia dviem žingsniais: 1) kontroliuojama šablono atitikimo baigtinės būsenos mašina; 2) teksto eilutė naudojama kaip įvestis automatui (Miretskiy *et al.* 2004).



2.6 pav. Reikšminių žodžių „abaa“ ir „abba“ {a,b} alfabete automatas. Sėkmingi perėjimai rodomi ištisinėmis linijomis. Galutinės būsenos rodomos paryškintais apskritimais. Nepavykę perėjimai parodomi punktyru (Miretskiy *et al.* 2004)

Fig. 2.6. An automaton for keywords “abaa” and “abba” over the alphabet fa,bg. Success transitions are shown with solid lines. Final states are shown with bold circles. Failure transition are shown with dotted lines (Miretskiy *et al.* 2004)

Šablonų sulyginimo automatui konstruoti Aho-Corasick algoritmas pirmiausia sukuria baigtinės būsenos mašiną visiems šablonams. 2.6 pav. parodytas reikšminių žodžių „abaa“ ir „abba“ {a,b} alfabete automatas. Būsena 0 žymi pradinę automato būseną, o galutinės būsenos parodomos paryškintais apskritimais.

Pirmiausia pridedamas šablonas „abaa“, kuris sukuria būsenas 0–4. Paskui pridedamas šablonas „abba“, sukuriantis 5–6 būsenas. Buvo reikalingos tik dvi papildomos būsenos, nes abu šablonai turi vienodą priešdėlį „ab“. Perėjimai šablono simboliais vadinami sėkmingais perėjimais (Miretskiy *et al.* 2004).

2.4. Antrojo skyriaus išvados

Skyriuje pristatyti kuriamos realiojo laiko skenavimo sistemos reikalavimai. Jie buvo suskirstyti į funkcinius, nefunkcinius ir detalizuoti.

Atrinkti ir detalizuoti šie funkciniai reikalavimai:

- Architektūriniai sistemos reikalavimai.
- Reikalavimai operacinėms sistemoms.
- Resursų naudojimo reikalavimai.
- Operatyviosios atminties skenavimo reikalavimas.
- Katalogų skenavimo reikalavimas.

Atrinkti ir detalizuoti šie nefunkciniai reikalavimai:

- Greitaveika.
- Darbo bei vartotojo sąsajos patogumas.
- Saugumas.

Pateiktas kuriamos sistemos projektas, pavaizduotas UML ir UMLSec diagramomis. Jose pateikiamos svarbiausios sistemos dalys, kurioms būtina skirti daugiau dėmesio. Naudojant UMLSec elementus suformuotos gairės tolesniems kūrimo proceso dalyviams. Gairės įpareigoja dalyvius laikytis nurodytų saugumo reikalavimų.

Skyriuje aprašyti darbai ir tyrimai leidžia pereiti į kitą sistemos kūrimo etapą. Jo metu turi būti eksperimento būdu apsispręsta dėl kitų sistemos dalių parinkimo, realizacijos bei laboratorinės aplinkos, reikalingos tyrimams ir rezultatų patikrai.

3

Realiojo laiko skenavimo sistemos įgyvendinimas ir efektyvumo analizė

Skyriuje aprašomos laboratorinės sąlygos, kuriomis buvo vykdomi tyrimai su KPK bei analizuojami vartotojų veiksmai dirbant su kompiuteriniais įrenginiais. Detaliai aptariama virtualios laboratorijos struktūra, aprašomi laboratorijos adaptacijos darbai ir pasiekti rezultatai.

Kituose poskyriuose aprašoma ekspertinė sistema, kuri yra kuriamos realiojo laiko skenavimo antivirusinės programos pagrindas, leidžiantis optimizuoti realiojo laiko skenavimą. Pasiekti rezultatai aptariami eksperimentinių tyrimų poskyryje, kuriame detaliai pateikiami tiek sistemos prototipo kūrimo technologijų bandymų rezultatai, tiek paties prototipo našumo ir palyginimo su kitais produktais rezultatai.

Skyriaus tematika paskelbti du autoriaus straipsniai (Radvilavičius *et al.* 2011; Čeponis, Radvilavičius 2008).

3.1. Ekspertinės sistemos aprašymas

Siūlomos ekspertinės sistemos architektūra

Ekspertinė sistema yra mūsų siūlomos sistemos viena pagrindinių ir bene svarbiausių dalių. Ją naudojant bus optimizuojamas vartotojo inicijuotų užklausų skenavimo realiuoju laikų skaičius. Mūsų siūloma ekspertinė sistema priskirtina taisyklėmis grįstų ekspertinių sistemų kategorijai, kurių pagrindą ir esmę sudaro eksperto sudarytų taisyklių rinkinys. Taip pat kai kurios duomenų bazės yra reikalingos vidinių duomenų ir parašų, naudojamų ClamAV varikliui, saugoti.

Main signature DB – duomenų bazė iš ClamAV – didžiausia kenkėjiško kodo duomenų bazė iš šio gamintojo. Ji atnaujinama kelis kartus per metus, kai visi parašai iš daily.cvd perkeliami į main.cvd duomenų bazę.

Daily signature DB – tai ClamAV duomenų bazė, kurioje saugomi naujausi kenkėjiško programinio kodo parašai. Ji atnaujinama kelis kartus per dieną. Kas kelis mėnesius visas jos turinys perkeliamas į main.cvd

File extension DB – pristatomo metodo vidinė duomenų bazė, kurioje saugomi skenuotinų ir neskanuotinų failų trumpiniai.

Standard file DB – duomenų bazė saugo standartinių operacinės sistemos failų kontrolines sumas, kurie pažymėti kaip patikimi ir niekuomet neturėtų būti skenuojami.

Internal hash DB – saugomos visų skenuotų failų kontrolinės sumos, kad antivirusinei programai nereikėtų to paties darbo daryti kelis sykius.

Most used files DB – kai analizuojama vartotojų elgsena ir saugomi daugiausia naudojami failai, turi būti duomenų bazė, kurioje būtų laikomi šie duomenys.

Ekspertinės sistemos taisyklių rinkinys ir jų pagrindimas

Taisyklėmis paremtas metodas susideda iš tam tikro taisyklių skaičiaus. Jos leidžia gauti reikalingą rezultatą – nuspręsti, ką skenuoti. Kiekvienas failas turi pereiti visas taisykles, kaip kad parodyta 2.7 pav., kad būtų leista prieiga prie failo, ar patikrinti jį su parašų duomenų bazėmis.

„Ar failo plėtinys yra extensions DB?“ taisyklė yra viena svarbiausių. Ji nusprendžia, ar failas turėtų būti skenuojamas pagal jo plėtinį. Taisyklė tampa vis svarbesnė, nes šiais laikais didžioji dalis grėsmių atkeliauja standartiniuose vykdomuosiuose failuose, kaip kad „exe“ ir t. t. Tokia pati taktika aprašyta JAV patente, pateiktame *Microsoft* (System and method of caching decisions on when to scan for malware) 2006 m. Jis aprašo, kaip failai turėtų būti parenkami skenuoti antivirusinės programinės įrangos, kai įvyksta skenuojamas įvykis, kaip kad I/O užklausa. Kitu atveju, jei failas nėra tokio tipo, kad galėtų būti infekuotas kenkėjiško kodo ir prieš tai nebuvo identifikuotas kaip užkrėstas, jis gali būti prieinamas be papildomo skenavimo.

„Ar failo dydis tinkamas skenavimu?“ taisyklė paremta failo dydžiu. Metodas nusprendžia leisti prieigą prie failo pagal failų dydį. Sistema turi derinamą failo dydį, kuris gali būti pakeistas norint pritaikyti naujoms kenkėjiškoms gijoms. Failo dydžio apribojimai veikiami kenkėjiškų programų specifikos – didžioji dalis kenkėjiškų programų nėra dideli failai. Ši taisyklė turi pavojingą pusę – kenkėjiškų programų kūrėjai gali sukurti failus, kurie būtų pakankamai dideli, kad nebūtų skenuojami, tad failų dydis turėtų būti atnaujinamas su parašų duomenų bazės atnaujinimais.

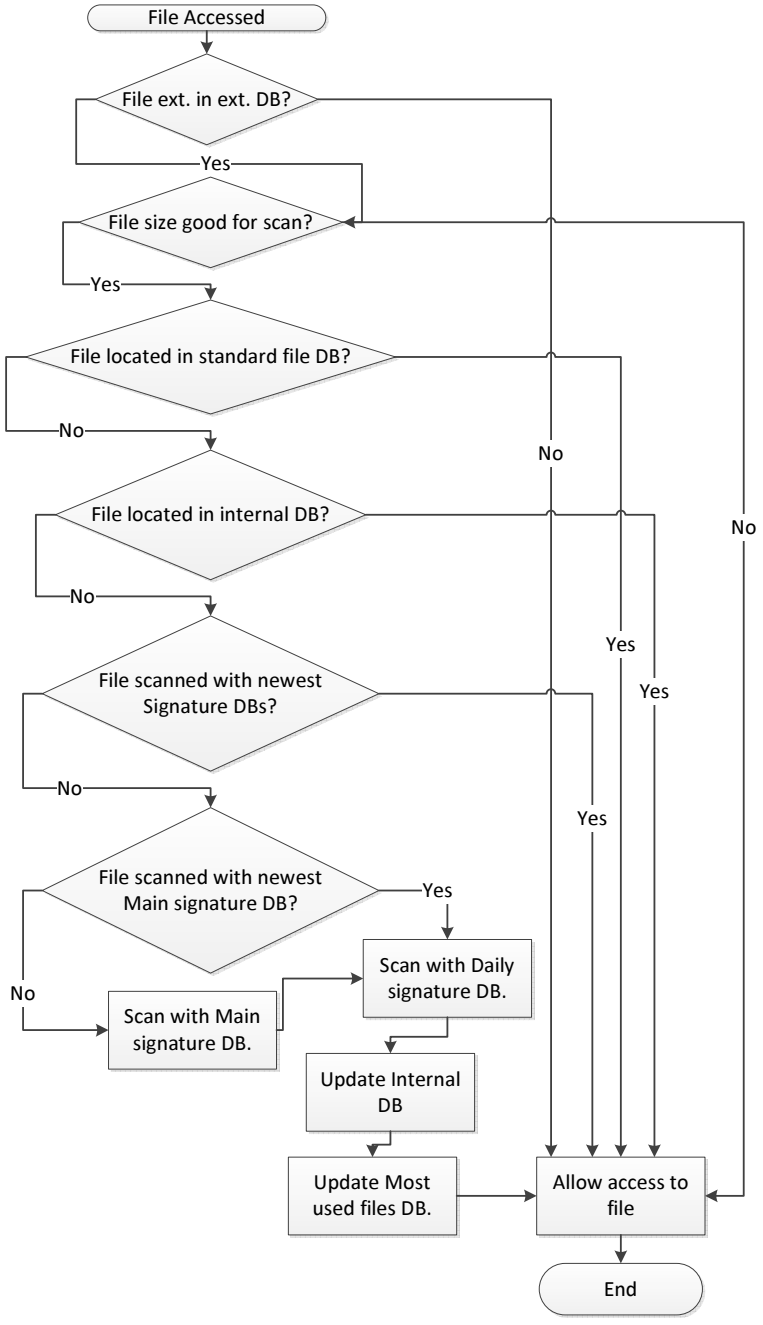
„Ar failas yra standard file DB“ – kaip ir visos operacinės sistemos bei programos (kaip kad *Microsoft Office*) ir paslaugos (pvz., duomenų bazių serverių) turi standartinius failus, kurie yra sukompiliuoti ir susieti gamintojų. Jie gali būti pažymimi kaip patikimi ir neskenuojami kiekvienąsyk. Ši taisyklė labai svarbi, nes didžiąją dalį laiko vartotojai dirba su standartiniais failais. Kai kurie šios taisyklės aspektai taip pat aprašyti JAV patente, pateiktame Kaspersky Lab 2010 m. (Method and system for antimalware… 2010): „Kai programinė įranga, žinoma antivirusinei (pvz., įdiegta *Microsoft Word* versija), paleidžiama, antivirusinės programos patikrinimas yra palyginti trumpas ir apribotas tik virusų parašų tikrinimui dinamiškai susietoms bibliotekoms. Kita vertus, kai programinė įranga, kuri nežinoma antivirusinei programai, yra paleidžiama pirmąsyk, atliekamas išsamus antivirusinės programos patikrinimas“.

„Failas, esantis internal DB“ – taisyklė patikrina, ar failas buvo skenuotas prieš tai ir su kuria nors parašų duomenų bazės versija. Tuo atveju, jei failas jau buvo skenuotas su naujausia duomenų baze, jis neskenuojamas darsyk, o metodas leidžia prieigą prie failo. Failui identifikuoti naudojamos maišos funkcijos.

Siūlomas metodas antivirusiniam patikrinimui diferencijuoja tarp žinomų vykdomų failų (t. y. tokių, su kuriais antivirusinė jau susidūrė tame pačiame kompiuteryje) ir nežinomų vykdomų failų (Method and system for antimalware… 2010).

„Failas skenuotas su naujausia parašų duomenų baze?“ – kai aišku, jog failas turi būti skenuojamas, svarbu sužinoti, ar jis buvo skenuojamas su dabartine parašų duomenų bazės versija. Jei buvo, tuomet tiesiog galima leisti prieigą prie failo neeikvojant brangaus laiko.

„Failas skenuotas su naujausia Main signature DB?“ – jei failas nebuvo skenuotas su naujausia parašų duomenų baze, dar svarbiau sužinoti, ar jis buvo skenuotas su pagrindine parašų duomenų baze. Tai svarbu todėl, kad pagrindinėje duomenų bazėje yra 846 214 parašų (2011 m. vasaris) (Schneier 1996), o kasdienėse duomenų bazėse būna tik 64 700 parašų (2011 m. vasaris),tai yra maždaug 13 kartų mažiau. Tai reiškia, kad 13 kartų mažiau naudojama CPU galios. Tuo atveju, jei failas buvo skenuojamas su parašais iš pagrindinės duomenų bazės, jis turi būti skenuojamas su naujausia parašų duomenų baze.



3.1 pav. Ekspertinės sistemos schema
Fig. 3.1. Scheme of expert system

Principinė metodo schema parodyta 3.1 pav. Yra keletas savybių, kurios buvo svarbiausios projektuojant ir kuriant sistemą. Viena jų buvo resursų išnaudojimo mažinimas. Kita – realiojo laiko skenavimo galimybių korekcija, kuri neleistų kenkėjiškam kodui užkrėsti kompiuterio.

3.2. Realiojo laiko skenavimo antivirusinės sistemos prototipas

Failų nuskaitymo tarnyba

Failų nuskaitymo tarnyba – programa-serveris, kuri veikimo metu laukia duomenų kopijavimo būdu siunčiamos informacijos ir ją persiunčia kanalų serveriui (Multithreaded Pipe Server 2010), 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 siunčiamieji duomenys. Eilutėje (4) gaunamas failo, kurį reikia tikrinti, adresas. (5) eilutėje failas siunčiamas tarnybai ir laukiama jos atsakymo. Jei rinkmena užkrėsta, grąžinama VIRUS reikšmė ((6) eilutė), jei rinkmena gera – grąžinama OK reikšmė ((7) eilutė).

Tarnyba pradeda veikti dar prieš vartotojui pradedant sesiją.Pradžioje ji įkelia virusų parašų duomenų bazes. Tuomet paleidžia kanalų serverį, kuris laukia susijungimo. Įvykus susijungimui, atlieka prašomą komandą ir grąžina atsakymą. Tarnyba darbą baigia, kai išjungiama operacinė sistema.

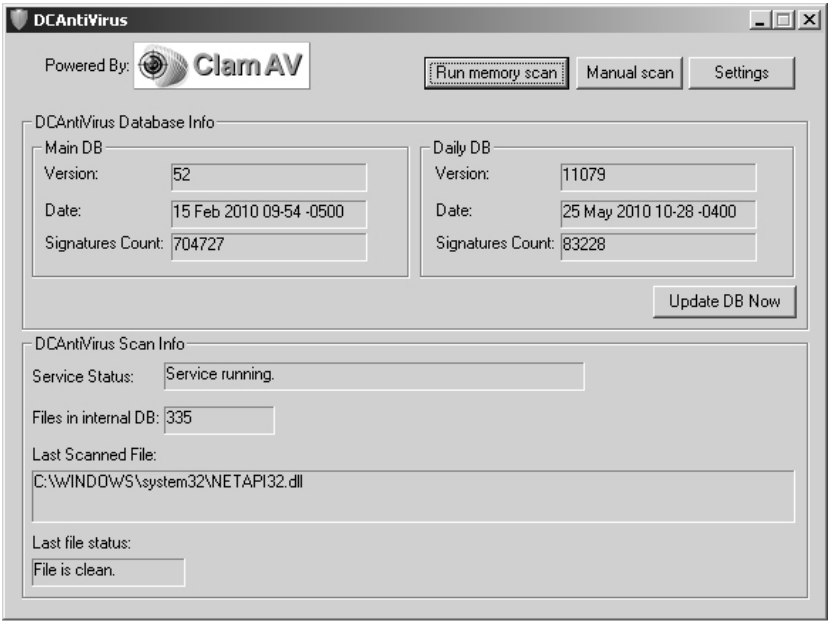
Sistemos dėkle esanti taikomoji programa

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

Pagrindinis programos langas

3.2 pav. pateikiamas pagrindinis programos langas.

Pagrindiniame lange pateikiama vartotojui aktuali informacija. Visų pirma pateikiama virusų duomenų bazių informacija (grupėje *DCAntiVirus Database info*). Rodomi pagrindinės ir kasdienės duomenų bazių numeriai, datos ir turimų parašų skaičius. Grupės apačioje yra mygtukas, skirtas duomenų bazėms atnaujinti.



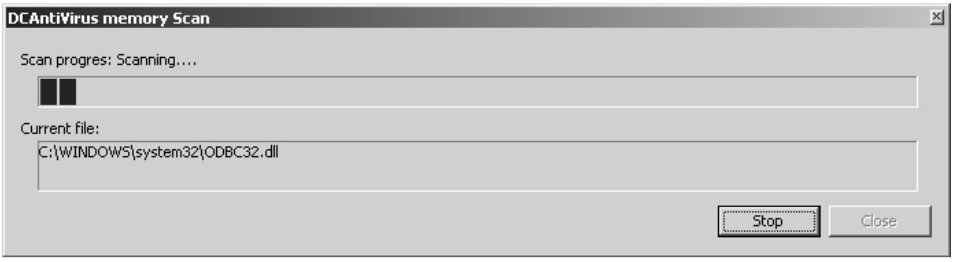
3.2 pav. Pagrindinis programos langas
Fig. 3.2. Main program window

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

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

Operatyviosios atminties nuskaitymas

3.3 pav. pateiktas operatyviosios atminties nuskaitymo langas. Jame rodoma nuskaitymo operacijos eiga ir šiuo metu nuskaitomas failas. Yra galimybė sustabdyti nuskaitymą. Baigus arba nutraukus nuskaitymą rodomas rezultatų langas (jis bus aptartas vėliau). Operatyviosios atminties nuskaitymas atliekamas taikant ToolHelp metodus (ToolHelp Functions 2010).

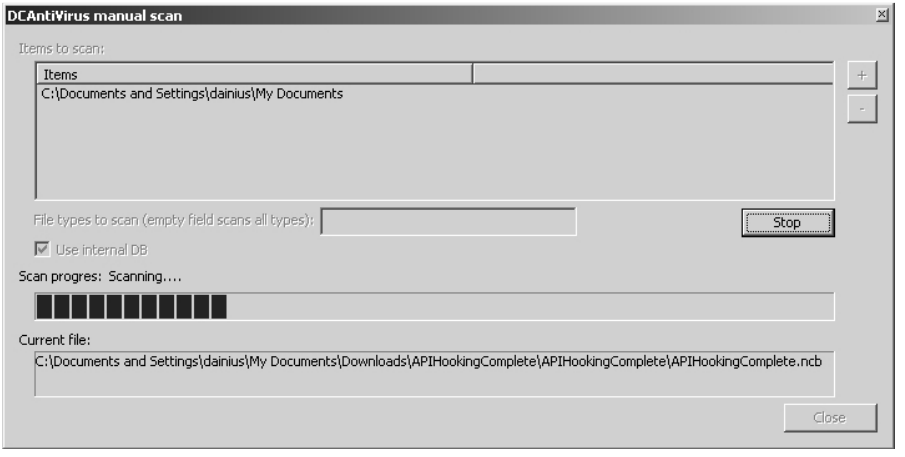


3.3 pav. Operatyviosios atminties nuskaitymas
Fig. 3.3. Scanning of operating memory

Rinkmenų ir katalogų nuskaitymas pagal pareikalavimą

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*.

3.4 pav. pateikiamas nuskaitymo pagal pareikalavimą eigos langas.



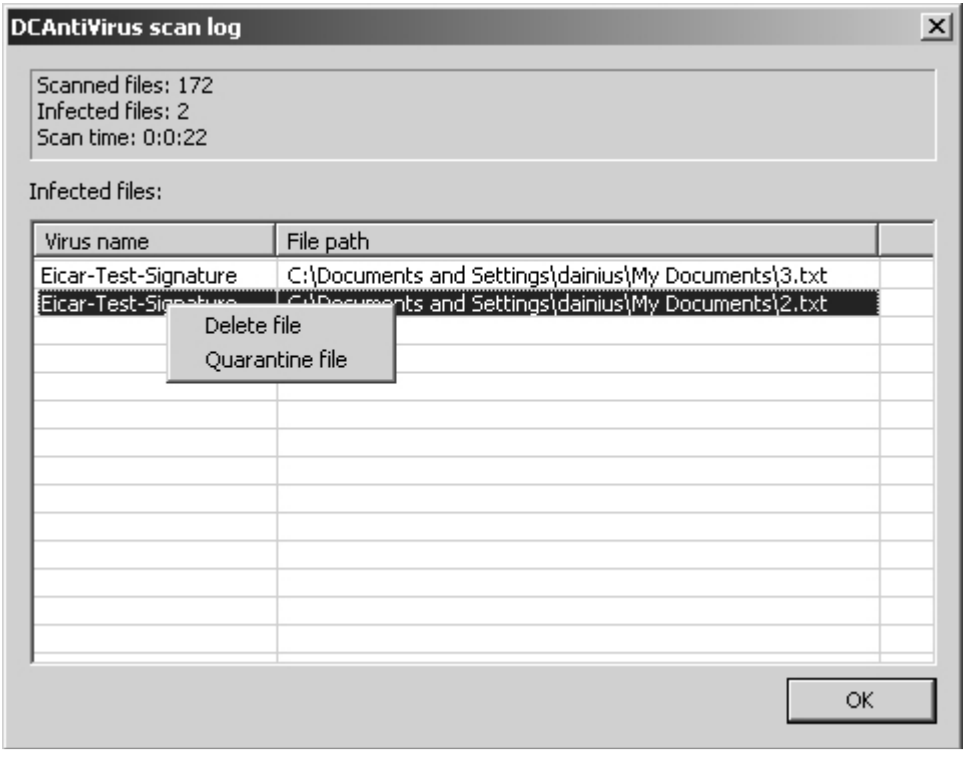
3.4 pav. Nuskaitymo pagal pareikalavimą eigos langas
Fig. 3.4. Scanning on demand window

Vykstant pasirinktų rinkmenų arba katalogų nuskaitymui, rodoma eiga ir šiuo metu nuskaitomas failas. Paspaudus mygtuką *Stop* arba pasibaigus nuskaitymu, rodomas rezultatų langas.

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 dešinįjį pelės klavišą atsiranda kontekstinis meniu (matomas 3.5 pav.). Jame galima pasirinkti du failo apdorojimo būdus:

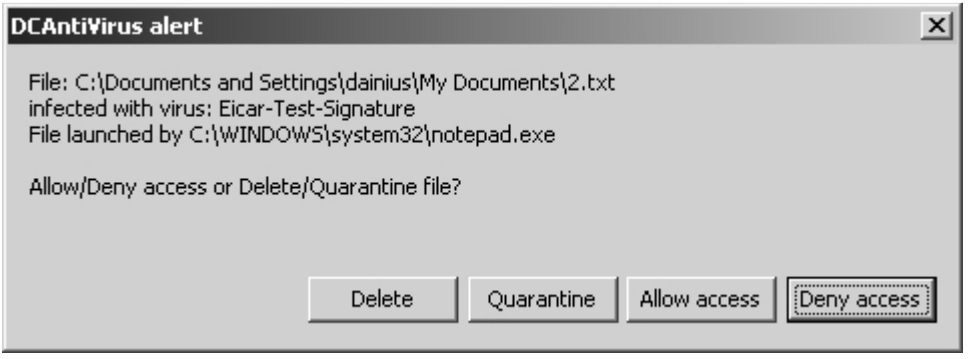
- *Delete file*. Rinkmena bus pašalinta iš standžiojo disko.
- *Quarantine file*. Rinkmena bus perkelta į karantino direktoriją.



3.5 pav. Nuskaitymo rezultatų langas
Fig. 3.5. Scan result window

Realiojo laiko nuskaitymo pranešimas apie užkrėstą rinkmeną

Realiojo nuskaitymo metu stebima visas sistema ir joje paliesti failai. Jei paliestas failas yra užkrėstas, apie tai pranešama vartotojui. Jam parodomas 3.6 pav. pateiktas dialogo langas.



3.6 pav. Pranešimas apie užkrėstą rinkmeną
Fig. 3.6. Message of virsu found

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

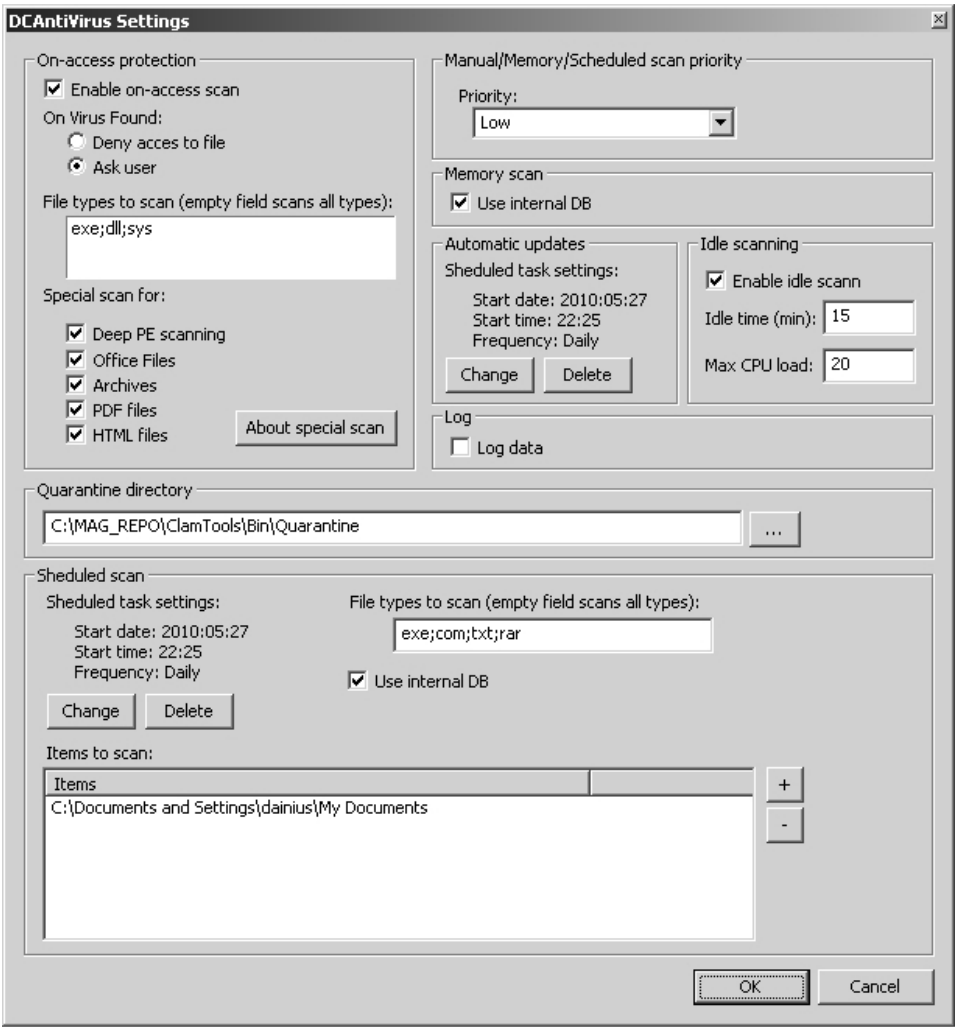
- *Delete.* Rinkmena bus ištrinta iš standžiojo disko.
- *Quarrantine.* Rinkmena bus perkeltaį karantino dialogą.
- *Allow access.* Leidžiama programai atidaryti norimą rinkmeną.
- *Deny access.* Draudžiama prieiti prie rinkmenos.

Programos nuostatų langas

Šiame lange (3.7 pav.) galima keisti programos nuostatas. Jos suskirstytos į logines grupes.

On-access protection. Šioje grupėje pateiktos realiojo laiko failų nuskaitymo nuostatos. Galima išjungti arba įjungti nuskaitymus, 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 papildomus failų nuskaitymo algoritmus.

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



3.7 pav. Programos nuostatų langas
Fig. 3.7. Settings window

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) (Task Scheduler 1.0 Interfaces 2010).

3.3. Laboratorinės aplinkos aprašymas

Norint išgauti teisingus rezultatus, reikalingos atitinkamos laboratorinės sąlygos. Tačiau, atsižvelgiant į kompiuterinės įrangos gamintojų gausą ir parametrų įvairovę, buvo apsiribota dešimčia darbo vietų su *Microsoft Windows* šeimos operacine sistema. Veiksmai buvo atliekami su KPK, todėl buvo būtinas lengvas darbo vietų atkūrimas po KPK pažeidimo bei išorinės aplinkos apsauga nuo galimo KPK plitimo. Tyrimui atlikti buvo pasirinkta virtuali laboratorinė aplinka, kuri detaliau aprašoma kituose poskyriuose.

Renkant duomenis buvo surinka daugiau negu 1 000 000 įrašų apie skirtingus vartotojo veiksmus. Duomenys rinkti nuo 2012 01 01 iki 2012 08 01.

Bazinė Telelab architektūra

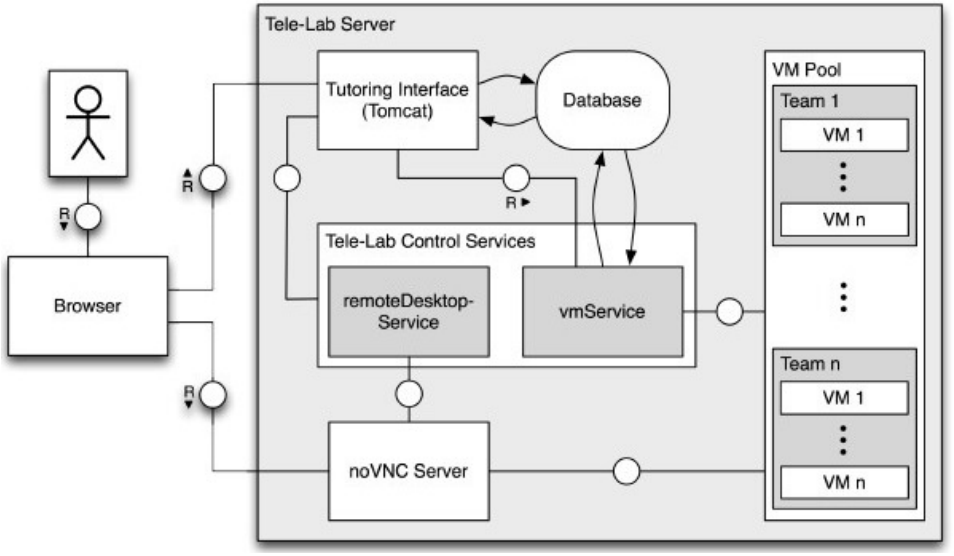
Projekto pradžioje Telelab buvo planuojama kaip atskira sistema (Hu *et al.* 2003), tačiau vėliau patobulinta iki virtualiosios laboratorijos sistemos, ėmus naudoti virtualias mašinas praktiniams pratimams atlikti (Hu, Meinel 2004), informacinių technologijų saugos studentams mokyti, ir toliau perkelta į Tele-Lab serverį (Hu *et al.* 2006; Willems, Meinel 2008). Tele-Lab serveris pateikia naujovišką e. mokymosi sistemą, padedančią praktiškai įsisavinti saugumą per WWW, ir perima visas gerąsias netinklinių mokymosi aplinkų charakteristikas.

Sistemos pagrindą sudaro virtualios mašinos ir jų valdymo platforma. Virtualios mašinos valdomos per nutolusią prieigą. Virtuali mašina yra programinės įrangos sistema, suteikianti veikimo aplinką operacinėms sistemoms. Tokios programinės įrangos emuliuojamos kompiuterinės sistemos, yra lengvai atstatomos ar perkeliamos įsivėlus klaidoms ar įvykus nesėkmėms.

Virtual Machine Pool: serveryje yra skirtingų virtualių mašinų aibė, kurios reikia praktinėms užduotims atlikti. Tai vadinama telkiniu. Fiziniai serverio resursai riboja maksimalų skaičių virtualių mašinų telkinyje. Praktikoje startuojamos kelios (3–5) mašinos. Jei visos komandos tam tikro pratimo scenarijui jau naudojamos, dinamiškai gali būti paleidžiamos naujos (tai vėlgi priklauso nuo fizinės serverio apkrovos). Šios mašinos dinamiškai sujungiamos į komandas ir priskiriamos vartotojui. Dabartinis valdymo sprendimas yra KVM/Qemu (Bellard 2011; Red Hat Inc. 2011). Libvirt paketas (The Libvirt Developers 2011) naudojamas kaip kevalas virtualios mašinos kontrolei. LVM (Linux Logical

Volume Management) pateikia virtualius kietuosius diskus, kurie gali imituoti *copy-on-write* diferencialines laikmenas. Tai svarbu siekiant išsaugoti vietą fiziniame kietajame diske, nes Tele-Lab serveris taip pat saugo virtualių mašinų šablonus kaip pagrindinius atvaizdus ir kiekvieną jų prireikus klonuoja pratimo aplinkoje. VM šablonuose taip pat yra konfigūraciniai failai, aprašantys techninius parametrus kaip kad atmintį, CPU skaičių ar tinklo sąsajas.

Tinklui sujungti tarp komandų Tele-Lab naudoja Virtual Distributed Ethernet (VDE) paketą (Davoli 2011). VDE emuliuoja visus fizinius Ethernet LAN aspektus programiškai. Tele-Lab Control Services paleidžia virtualius perjungiklius ar skirstytuvus kiekvienam virtualiam tinklui, apibrėžtam VM komandai, ir sujungia mašinas į atitinkamą tinklo infrastruktūrą. Norint paskirstyti IP adresus virtualiuose tinkluose prie kiekvieno tinklo prijungiamas DHCP serveris. Išsiuntus visus adresus, DHCP serveris išjungiamas saugumo sumetimais.



3.8 pav. Tele-Lab sistemos architektūros apžvalga
Fig. 3.8. Overview – Architecture of the Tele-Lab Platform

Remote Desktop Access Proxy: Tele-Lab serveris turi susidoroti su lygiagrečiaisiais nutolusios prieigos susijungimais, kai vartotojai naudojasi sistema. Tai įgyvendinta naudojant atvirojo kodo projektą noVNC, kurio klientas, skirtas Virtual Network Computing protokolui, yra paremtas HTML5 Canvas ir WebSockets (Martin 2011) noVNC pakete yra HTML5 klientas ir WebSockets įgaliotasis serveris, kuris sujungia klientus ir VNC serverius, pateikiamus QEMU. Užtikrinti apsaugotą aplinką tiek Tele-Lab vartotojams, tiek sistemai

yra sudėtinga užduotis ir ją svarbu korektiškai įgyvendinti visais lygmenimis, nes tinklo saugumo reikalavimai virtualioms mašinoms debesų kompiuterių nustatymuose (kaip kad yra su Tele-Lab) turi specifinių reikalavimų. Sistema naudoja žymėmis paremtą tapatumo nustatymo sistemą: prieigos žymė nutolusiam prisijungimui sugeneruojama, kai vartotojas pareikalauja virtualios mašinos komandos atlikti pratimą. TLS naudojimas užtikrina žymės konfidencialumą.

Tele-Lab Control Services: centrinių Tele-lab kontrolės tarnybų tikslas yra visų minėtų komponentų sujungimas. Abstrakcijos lygmeniui realizuoti, skirtam virtualios mašinos stebėtojui (ar valdytojui) ir nutolusio prisijungimo įgaliotajam serveriui sujungti, sistema įdiegia lengvų XML-RPC interneto tarnybų rinkinį: vmService ir remoteDesktopService. vmService naudojamas virtualių mašinų kontrolei – startuoti, stabdyti ar atstatyti, grupuoti į komandas ar priskirti mašinas bei komandas vartotojui. remoteDesktopService naudojamas inicializuoti, startuoti, stebėti ir nutolusiem susijungimam į mašiną, priskirtą studentui pratimams atlikti, užbaigti. Minėtos Grails programos (portalas, mokymosi aplinka ir administracinė prieiga) leidžia vartotojui ir administratoriui kontroliuoti visą sistemą, naudojant interneto tarnybas.

Kliento pusėje vartotojui reikia tik naršyklės, palaikančios SSL/TLS. Dabartinė noVNC kliento implementacija nereikalauja net HTML5 palaikančios naršyklės: senesnėms naršyklėms, HTML5 Canvas ir/arba WebSockets emuliuojami naudojant Adobe Flash programinį paketą.

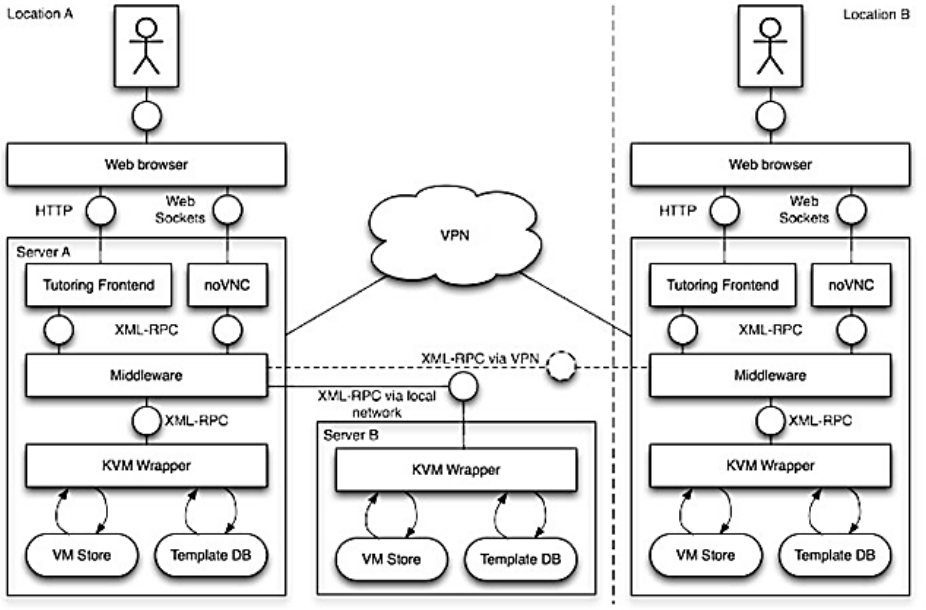
Paskirstytosios Telelab architektūros naudojimas tinkliniam KPK aptikti ir analizuoti

Vien tik KPK analizuoti ir vartotojo veiksmams fiksuoti visiškai pakaktų nuo išorinio pasaulio atskirtos operacinės sistemos, veikiančios virtualioje mašinoje. Tačiau, kaip minėta apžvalgoje, šiais laikais kenksmingas programinis kodas yra prisitaikęs prie tinklinių struktūrų ir plinta įvairiais pavidalais bei protokolais. Dėl to, naudojant laboratorinę aplinką, savarankišką virtualią mašiną su operacine sistema bei vartotojo naudojamomis taikomosiomis programomis, tačiau be ryšio su kitų vartotojų kompiuteriais ir kitomis tinklinėmis prieigomis, būtų sukurtas imitacinis modelis, stipriai atsiliekantis nuo šios dienos realių IT sistemų.

Dėl šių priežasčių KPK analizei buvo pasirinkta virtualios laboratorijos sistema Tele-Lab, ji išplečiama ir pritaikomas mūsų poreikiams. Kadangi būtina imituoti kuo realesnį kompiuterio vartotojo darbą, buvo būtinos ne tik pavienės virtualios mašinos, bet ir komunikacija tarp jų, imituojant tiek vietinį organizacijos tinklą, tiek internetą. Šiai užduočiai Tele-Lab sistema buvo patobulinta iki tinklinės virtualios laboratorijos, o virtualios mašinos turėjo bendrauti ne tik su kitomis virtualiomis mašinomis, dirbančiomis su tuo pačiu serveriu, bet ir nutolusiomis, fiziškai esančiomis kitame serveryje.

Laboratorinės aplinkos saugumo užtikrinimas

Kadangi Tele-Lab pateikia vartotojams virtualias mašinas su visomis privilegijomis, jos turėtų būti naudojamos mokytis analizuoti KPK, sistemos saugumas yra itin svarbus įdiegimo klausimas. Bendruoju atveju, implementacija siekia sumažinti atakos paviršių ir leisti kuo mažiau atakos vektorių: vienintelės paslaugos, prieinamos iš interneto, turėtų būti interneto serveris mokymosi sąsajai (80 ir 443 prievadai) ir noVNC įgaliotasis serveris nutolusiam susijungimui (10099 prievadas). Net interneto sąsaja, skirta administruoti, prieinama tik iš vidinio tinklo, XML-RPC paremtos paslaugos yra prieinamos tik iš to paties serverio arba (klasterizavimo prielaida, pavaizduota lokacijoje A, 3.9 pav.) iš kitų Tele-Lab serverių vietiniame tinkle perspektyvos.



3.9 pav. Tele-Lab serverių klasterizacija: vietinis klasterizavimas (A lokacija) ir atskirų atvejų klasterizavimas

Fig. 3.9. Clustering of Tele-Lab Servers: On-Site Clustering (Location A) and Clustering of Independent Instances

VNC nutolusiam susijungimui skirti prievadai neprieinami iš interneto – XML-RPC paslaugų politika taip pat taikoma ir šiai paslaugai. Visi nutolusios prieigos susijungimai turi būti inicijuojami per noVNC įgaliotąjį serverį, kuris patikrina perduodamas prieigos žymas iš užklausos duomenų bazei ir taip įvertina, ar prieiga gali būti autorizuota.

Apsisaugojimui nuo IP klastojimo atakų (išoriniai kenkėjai modifikuoja IP adresus, kad šie atrodytų priklausantys vidiniam tinklui) yra papildoma fizinė tinklo sąsaja, reikalinga komunikacijai su vietiniu tinklu. Visi paketai, siunčiami iš vietinių IP adresų ir gaunami tinklo sąsajoje, skirtoje susijungti su internetu, atmetami ugniasienės.

Patobulinta sistema reikalauja prieigos prie XML-RPC paslaugos, suteikiamos vidurinio lygmens ne toje pačioje lokacijoje esančioms Tele-Lab sistemoms, t. y. Tele-Lab serveris Lietuvoje turi galėti prieiti prie vidurinio lygmens paslaugų, pateikiamų Tele-Lab serverio, esančio, pvz., Vokietijoje. Prieiga prie vidurinio lygmens paslaugų leidžia įvairias atakas, kaip kad didelio kiekio virtualių mašinų startavimas ir perkrovos sukūrimas ar VM komandų, kurios naudojamos, išjungimas. Todėl reikalingi stiprūs abipusiai tapatumo nustatymo ir autorizacijos mechanizmai.

Pakankamas sprendimas gali būti pateiktas naudojant virtualų privatų tinklą. Tele-Lab serveriai sujungti VPN tuneliu, naudojančiu Ipsec ir sertifikatus. Praktikoje OpenSWAN sukonfigūruotas VPN tuneliui sukurti. OpenSWAN sukuria naują tinklo sąsają abiejuose kompiuteriuose skirtinguose VPN tunelio galuose. Ugniasienė turi būti pat priimti užklausas į XML-RPC vidurinio lygmens prievadą iš šių tinklo sąsajų. Kadangi VPN taip pat užšifruoja visą srautą, tai yra papildoma apsauga prieš atkartojimo stiliaus atakas.

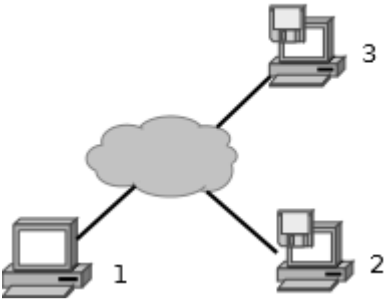
Ateityje šio sprendimo išplėtimui pagerinti svarstomos saugumo žymos prieigai prie vidurinio lygmens paslaugų.

Tele-Lab naudojimo išplėtimas antivirusinei sistemai testuoti

Įdiegus Tele-Lab Vilniaus Gedimino technikos universitete, buvo vykdomas išplėstinis virtualių serverių ir veikimo metodų testavimas. Viena svarbiausių užduočių – pamėginti ir adaptuoti Tele-lab platformą ne tik mokymosi užduotims, bet išplėsti ir patį naudojimo mastą.

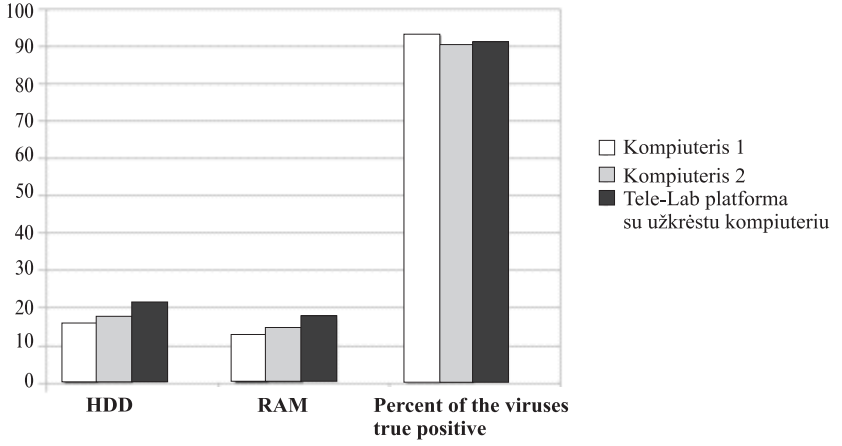
Viena svarbiausių realiojo laiko skenerio užduočių – apsaugoti vartotoją nuo duomenų srauto iš tinklo. Preliminari trijų virtualių mašinų schema pavaizduota 3.10 pav. Aukos kompiuteris su *Microsoft Windows XP* operacine sistema ir įdiegta realiojo laiko taisyklėmis paremta antivirusine sistema yra pažymėtas Nr. 1. Jis stebi duomenų srautą, ateinantį iš vietinio tinklo. Toje pačioje komandoje įgyvendinti du atakuojantys kompiuteriai (pažymėti atitinkamai Nr. 2 ir 3).

Atakuojantys kompiuteriai paruošti su dinaminiais IP adresais, tad kaskart jiems mėginant komunikuoti su aukos kompiuteriu, jų IP adresai skiriasi, todėl simuliuojamas didesnio tinklo modelis. Tai vykdoma dėl Tele-Lab apribojimų, dėl kurių komandos gali būti formuojamos tik iš riboto kompiuterių skaičiaus. Atakuojantysis inicijuoja susijungimą su aukos mašina ir jai mėgina nusiųsti kenksmingą kodą. Kenksmingas kodas paimamas iš 0day clamAV antivirusinės duomenų bazės (testavimo dienos metu).



3.10 pav. Testavimo aplinkos architektūros schema
Fig. 3.10. Scheme of the testing environment architecture

Po kiekvieno inicijuoto susijungimo atakuojančiojo sistema pakeičia IP adresą, pašalina mėgintą kenksmingą programinį kodą iš sąrašo, laukia 30 sekundžių ir vėl mėgina jungtis prie aukos kompiuterio.



3.11 pav. Sistemos apkrovimo ir efektyvumo palyginimas (HDD, RAM tikrinti su HWMonitor)
Fig. 3.11. Comparison of system load and efficiency (HDD, RAM checked with HWMonitor)

Taisyklėmis paremtos realiojo laiko antivirusinės sistemos supaprastintame žurnaliniame įraše įmanoma patikrinti antivirusinės sistemos efektyvumą, greitį ir aukos kompiuterio darbo apkrovą. Darbo apkrova ir antivirusinės sistemos efektyvumas labai panašus į realiojo pasaulio skaičius beveik identiškose sistemose kaip, kad parodyta 3.11 pav. Tai leidžia panaudoti Tele-Lab platformą

įvairiems automatiniams programinės įrangos patikrinimams ir praplečia platformos galimybes.

3.4. Eksperimentinių tyrimų rezultatai

Procesų bendradarbiavimo greičio tyrimo rezultatai

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

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

*Bendravimas naudojant kanalu*s (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 (Interprocess Communications 2010).

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 (Interprocess Communications 2010).

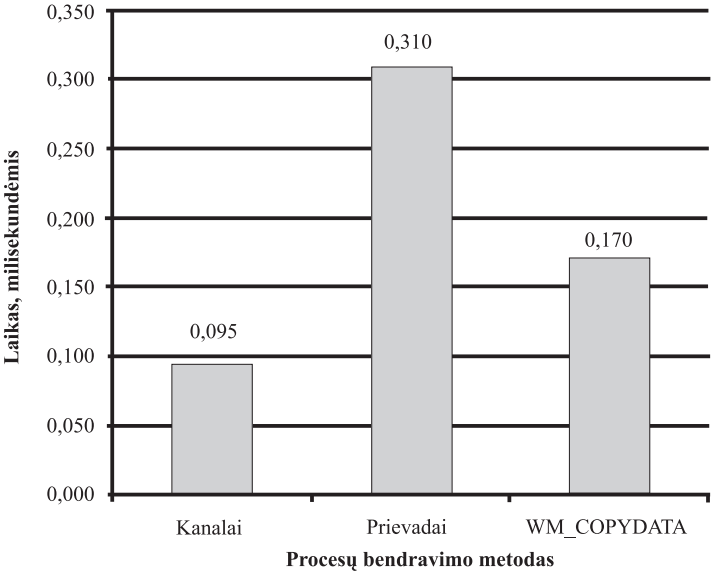
Bendravimas kopijuojant duomenis (angl. *Data Copy*). Šis metodas leidžia iš vienos programos nusiųsti informaciją kitai programai naudojant WM_COPYDATA žinutę. Programa-serveris turi tiksliai žinoti gaunamos informacijos struktūrą. Programa-klientas privalo nemodifikuoti siunčiamos informacijos, kol ją apdoroja programa-serveris. Norint pasiųsti duomenų kopijavimo žinutę, reikia žinoti programos-gavėjo pavadinimą (Interprocess Communications 2010).

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 programa 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 ir atsakymo laukimas. Programa-serveris atsakymą išsiųsdavo vos tik gavęs duomenis. Kiekvienas testas atliktas 10 kartų. Testo rezultatai pateikiami 3.12 pav.



3.12 pav. Procesų bendravimo metodų testo rezultatai
Fig. 3.12. Test results of process communication methods

Pagal testo rezultatus funkcijos darbo spartą mažiausiai veikė 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.

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

Taigi atsižvelgiant į testo rezultatus ir 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* ir *Windows 7*) duomenų kopijavimo metodą įgyvendinti tarnyboje neįmanoma (Interactive Services 2010). *Microsoft*, siekdama padidinti sistemos saugumą, pradėjo griežtai kontroliuoti tarnybų darbą ir jas pateikė paprastoms taikomosioms programoms neprieinamme darbalaukyje (Security, services and the interactive… 2005).

Bylų maišos funkcijų tyrimas

Bylų nuskaitymo modulis naudojamas rinkmenai patikrinti. Failas tikrinamas naudojant ClamAV virusų parašų duomenų bazes. Jos yra dvi. Tai pagrindinė (atnaujinama kartą per savaitę) ir kasdienė (atnaujinama kiekvieną dieną). Siekiant visiško virusų aptikimo tikslumo, naudojama ClamAV siūloma biblioteka, kuri turi rinkmenos nuskaitymo funkciją. Ši biblioteka (libclamav.dll) platinama nemokamai. Biblioteka taiko naujausius metodus ir algoritmus virusų paieškai (Miretskiy *et al.* 2004). Keli iš jų:

- Aho–Corasick eilutės paieškos algoritmas (Miretskiy *et al.* 2004).
- Išplėstas Boyer–Moore eilutės paieškos algoritmas (Miretskiy *et al.* 2004).

ClamAV bibliotekos palaikomieji rinkmenų formatai:

- Vykdomieji failai.
- Elektroninio pašto rinkmenos.
- Populiarūs archyvo formatai.
- Dokumentai (Microsoft Office, PDF, HTML)
- 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);
```

Funkcijai perduodamas rinkmenos adresas (filename), rodyklė, į kurią įrašomas viruso pavadinimas (virname), ir 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;
}
```

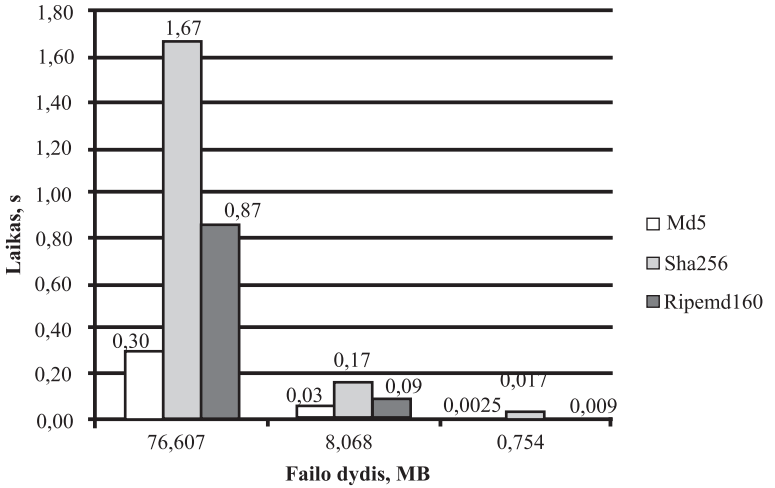
Eilutėje (1) patikrinama, ar rinkmenos tipas atitinka vartotojo nustatytas nuskaitymui rinkmenas. Eilutėje (2) patikrinama, ar rinkmena nėra išsaugota kaip neužkrėsta. Jei ji egzistuoja vidinėje duomenų bazėje, patikrinama, kokie buvo main ir daily duomenų bazių numeriai. Aptikus kurį nors nesutapimą – rinkmena nuskaitoma su nesutapusia virusų parašų duomenų baze. Tokiu būdu užtikrinamas tik reikiamas nuskaitymas. Tuomet (3) ir (4) eilutėse rinkmena nuskaitoma. 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 (Schneier 1996). Tam atlikti yra daugybė būdų. Tačiau reikia pasirinkti sparčiausią, nes kitu atveju failo atidarymas gali užtrukti ne tik dėl nuskaitymo, bet ir dėl kontrolinės sumos suskaičiavimo (Marshall *et al.* 1995). Kontrolinės sumai suskaičiuoti 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 programa, kuri su kiekvienu metodu suskaičiuodavo trijų skirtingo dydžio rinkmenų kontrolines sumas. Kiekvienas suskaičiavimas kartotas 10 kartų, testų rezultatuose pateikiamas jų vidurkis.

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

Table 3.1. Test results of hash function usage. Time is presented in seconds

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



3.13 pav. Kontrolinės sumos suskaičiavimo testo grafinė išraiška
Fig. 3.13. Test results of hash function usage

Atlikus testą paaiškėjo, kad kontrolinę sumą sparčiausiai suskaičiuodavo MD5 algoritmas. Žinoma, visų algoritmų sparta sumažėdavo dirbant su dideliu failu. Bet visais atvejais MD5 būdavo tris 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 ir paėmimą. Vidinė duomenų bazė saugo failo adreso kontrolinę suma, failo kontrolinę sumą, daily ir main duomenų bazių numerius, failo naudojimo skaitiklį ir 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 užblokuojamas. Taip išvengiama failo sugadinimo arba duomenų pakeitimo. Failas užblokuojamas naudojant Windows API funkciją LockFile(...).

Ooperacinių sistemų sisteminių užklausų perėmimo metodų našumo charakteristikos vertinimas

Dvi realiojo laiko skenavimo variklio dalys yra svarbiausios apibrėžiant variklio našumą: procesų pagavimo laikas ir failo skenavimo laikas. Šiame skyriuje bus skirta dėmesio pagavimo bibliotekų laiko analizei ir failų skenavimo laikui.

Norint perimti *Windows* API funkcijas reikia gerai išmanyti operacinę sistemą, nes toks procesas siejamas su sistemos atminties procesų modifikavimu. Naudojamų bibliotekų pasirinkimas priklauso nuo reikalavimų. Pasirenkamas metodas realiajam laikui skenuoti. Metodas turėtų gebėti perimti funkcijas, kurias operacinė sistema naudoja darbui su failais, t. y. *Kernel32.dll* funkcijas *CreateFile*, *OpenFile*, *ReadFile* ir t. t. Taip pat funkcionuojančios sistemos greitis turi būti minimizuotas, o bibliotekos turi būti parašytos GPL. Šiame teste pasirinktos ir naudojamos Detours ir EasyHook bibliotekos.

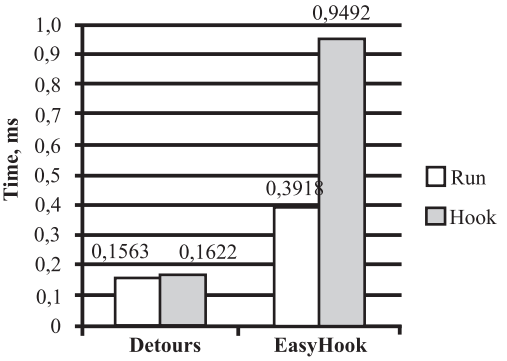
Detours biblioteka perima funkcijas, perrašydama procesų iškvietimo lenteles. Kiekvienai funkcijai biblioteka perrašo dvi funkcijas. Detours biblioteka gali perimti bet kokios bibliotekos bet kokią funkciją.

Easyhook biblioteka palaiko nevaldomo išeigos kodo perėmimą, nuskaitydama valdomą kodą. Ji užtikrina, kad jokių resursų ar atminties liekanų perimtoje programoje nelieka. Taip pat ji geba naudoti manipuliavimo programas nekontroliuojamoms API funkcijoms perimti.

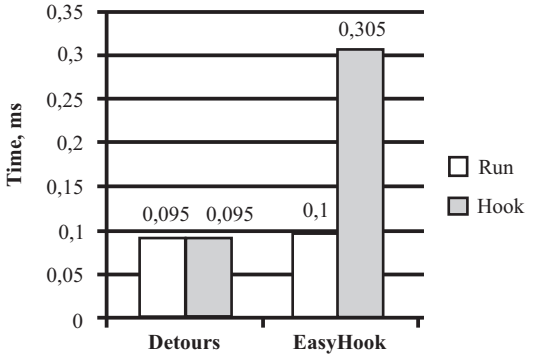
Viena svarbiausių siūlomo metodo savybių yra greitis, todėl šie testai yra itin svarbūs nustatant, kurią biblioteką naudoti. Korektiškas bibliotekos pasirinkimas veikia bendrą sistemos greitį.

Kiekvienai bibliotekų buvo sukurtos programos-serveriai. Jie palaiko komandinės eilutės formatą. Programos neatlieka jokių užduočių, tad laikas nėra veikiamas. *Run* – programa įkraunama ir laukia, kol bus užbaigtas naujas procesas. *Hook* – viena bibliotekų įterpiama ir perima funkcijos *CreateFile* kvietimą.

Iš 3.14 pav. aišku, kad kai programa startuojama su EasyHook, ji tai daro kur kas lėčiau (142,27 %). 3.18 pav. pateikiamas bibliotekos darbo testas. Galima pastebėti, kad Detours biblioteka neveikia veikimo laiko, o EasyHook vėlgi rodo prastesnius rezultatus. Ji lėtesnė 305 %.



3.14 pav. Bibliotekų įkrovimo laikas
Fig. 3.14. Libraries load test



3.15 pav. Bibliotekų darbo testas
Fig. 3.15. Libraries running test

Ekspertinės sistemos našumo eksperimentinis patikrinimas

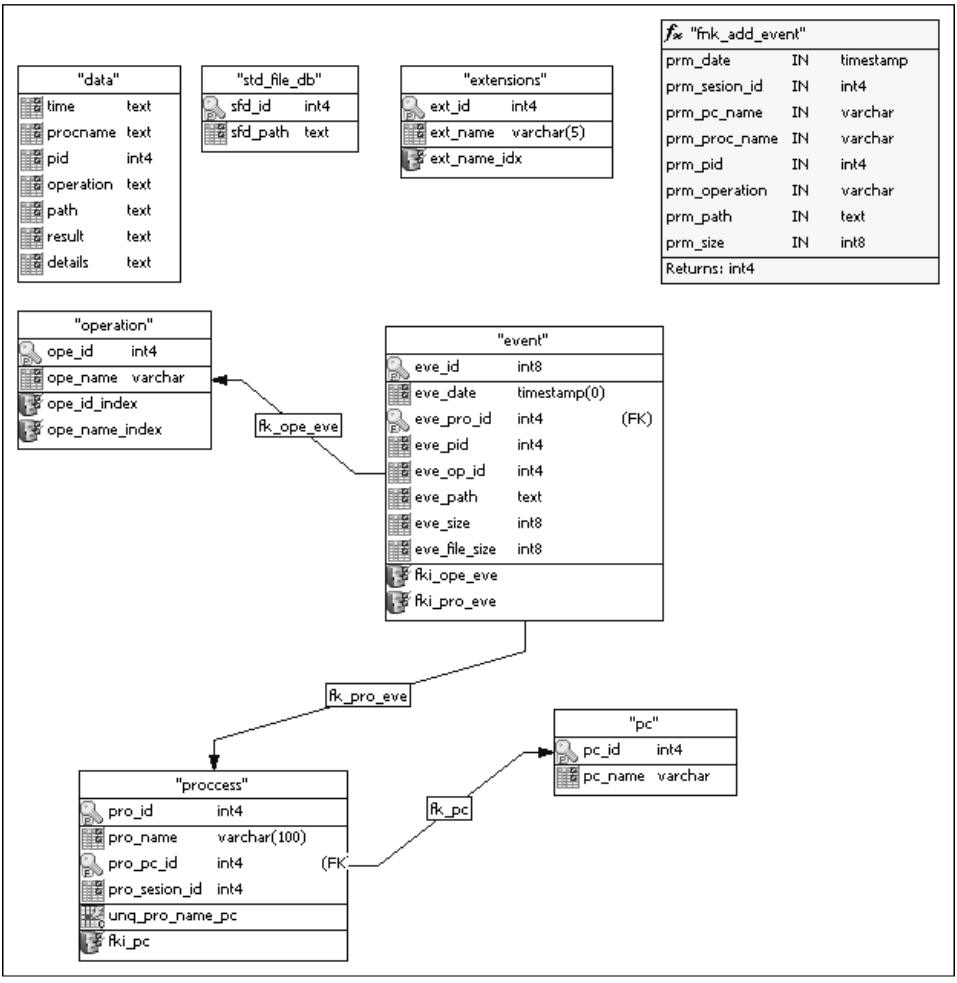
Kadangi ekspertinei sistemai buvo keliama pagrindinė užduotis – sumažinti tikrinimui perduodamų užklausų skaičius, o užklausų įvairovė yra be galo didelė, nes priklauso nuo vartotojo veiksmų, jos patikimumo ir našumo patikrinimas galimas tik su dideliu kiekiu kuo įvairesnių duomenų. Našumo patikrinimo eksperimentas buvo suskirstytas į šias dalis:

- Vartotojo veiksmų stebėjimas ir duomenų surinkimas.
- Duomenų išsaugojimas releacinėje duomenų bazėje.
- Duomenų apdorojimas ir rezultatų suskaičiavimas.

Vartotojo veiksmų stebėjimo metodas jau buvo detaliai aprašytas, jį taikant buvo renkama informacija apie bet kokio vartotojo inicijuoto proceso kreipimąsi į failų sistemą. Prie veiksmo buvo priskiriami papildomi parametrai, tokie kaip bylos pavadinimas, dydis, veiksmo tipas, data, laikas ir pan.

Duomenims išsaugoti buvo parinkta PostreSQL duomenų bazių valdymo sistema. Duomenys saugomi specialiai tam sukurtoje duomenų bazėje, kurios schema pateikiama 3.16 pav.

Duomenų bazė buvo sudaryta iš septynių lentelių, vienos duomenų bazės funkcijos skirtos korektiškam duomenų įvedimui. Kuriant duomenų bazę buvo laikomasi taisyklės, kad kiekviena iš lentelių turi po sveikojo tipo (angl. *Integer*) pirminį raktą (angl. *Primary key*), kuris duomenų bazės valdymo sistemos yra generuojamas iš eilės didėjimo tvarka. Lentelės, turinčios ryšį su kitomis lentelėmis, turi po sveikojo skaičiaus tipo stulpelį su išoriniu raktu (angl. *Foreign Key*), kuris atitinkamai jungiamas su pirminiu kitos lentelės raktu.



3.16 pav. Duomenų bazės schema
Fig. 3.16. Datebase scheme

Toliau pateikiamas vienos pagrindinių duomenų bazės lentelių išsamus aprašymas bei visų kitų aprašymo santraukos.

Duomenų lentelė „Event“
Aprašymas: duomenų lentelėje saugomi agreguoti duomenys apie naudotojų kompiuteriuose atliktus veiksmus. Ši lentelė susieta su kitomis išoriniais raktais. Vienas išorinių raktų rodo duomenų bazės lentele „operation“, kurioje saugomi operacijų pavadinimai. Kitas nurodo į „proccess“ lentelę, kurioje saugomi procesų pavadinimai.

```
CREATE TABLE "event" (
                "eve_id" BIGSERIAL NOT NULL,
                "eve_date" timestamp(0),
                "eve_pro_id" int4 NOT NULL,
                "eve_pid" int4,
                "eve_op_id" int4,
                "eve_path" text,
                "eve_size" int8,
                "eve_file_size" int8,
 CONSTRAINT "event_pkey" PRIMARY KEY("eve_id"),
 CONSTRAINT "fk_ope_eve" FOREIGN KEY ("eve_op_id")
  REFERENCES "operation"("ope_id")
                MATCH SIMPLE
                ON DELETE CASCADE
                ON UPDATE CASCADE
                NOT DEFERRABLE,
 CONSTRAINT "fk_pro_eve" FOREIGN KEY ("eve_pro_id")
 REFERENCES "proccess"("pro_id")
                MATCH SIMPLE
                ON DELETE CASCADE
                ON UPDATE CASCADE
                NOT DEFERRABLE
)
WITHOUT OIDS;
CREATE INDEX "fki_ope_eve" ON "event" USING BTREE (
                "eve_op_id"
);
CREATE INDEX "fki_pro_eve" ON "event" USING BTREE (
                "eve_pro_id"
);
```

3.2 lentelė. Duomenų lentelė „Event“
Table 3.2. Database tabale „Event“

Column Name	Data Type	Primary Key	Not Null	AutoInc	Komentarai
eve_id	int8	YES	YES	YES	
eve_date	timestamp(0)	NO	NO	NO	
eve_pro_id	int4	NO	YES	NO	
eve_pid	int4	NO	NO	NO	
eve_op_id	int4	NO	NO	NO	
eve_path	text	NO	NO	NO	
eve_size	int8	NO	NO	NO	
eve_file_size	int8	NO	NO	NO	in bytes

Lentelės „event“ kūrimo SQL kode pavaizduota visa lentelės schema kartu su vienu pirminiu raktu, dviem išoriniais raktais ir dviem indeksais su visais parametrais.

Duomenų lentelė „Extensions“
Aprašymas: duomenų lentelėje saugomi bylų plėtinių pavadinimai.

3.3 lentelė. Duomenų lentelė „Extensions“
Table 3.3. Database tabale „Extensions“

Column Name	Data Type	Primary Key	Not Null	AutoInc
ext_id	int4	YES	YES	YES
ext_name	varchar(5)	NO	NO	NO

Duomenų lentelė „Operation“
Aprašymas: duomenų lentelėje saugomi operacijų su bylomis pavadinimai.

3.4 lentelė. Duomenų lentelė „Operation“
Table 3.4. Database tabale „Operation“

Column Name	Data Type	Primary Key	Not Null	AutoInc
ope_id	int4	YES	YES	YES
ope_name	varchar	NO	NO	NO

Duomenų lentelė „Pc“
Aprašymas: Duomenų lentelėje saugomi vartotojų kompiuterių pavadinimai.

3.5 lentelė. Duomenų lentelė „Pc“
Table 3.5. Database tabale „Pc“

Column Name	Data Type	Primary Key	Not Null	AutoInc
pc_id	int4	YES	YES	YES
pc_name	varchar	NO	NO	NO

Duomenų lentelė „Proccess“
Aprašymas: duomenų lentelėje saugomi procesų pavadinimai ir sesijų identifikatoriai.

3.6 lentelė. Duomenų lentelė „Proccess“
Table 3.6. Database tabale „Proccess“

Column Name	Data Type	Primary Key	Not Null	AutoInc
pro_id	int4	YES	YES	YES
pro_name	varchar(100)	NO	NO	NO
pro_pc_id	int4	NO	NO	NO
pro_sesion_id	int4	NO	NO	NO

Duomenų lentelė „std_file_db“
Aprašymas: duomenų lentelėje saugoma standartinių bylų duomenų bazės informacija.

3.7 lentelė. Duomenų lentelė „std_file_db“
Table 3.7. Database tabale „std_file_db“

Column Name	Data Type	Primary Key	Not Null	AutoInc
sfd_id	int4	YES	YES	YES
sfd_path	text	NO	NO	NO

Duomenų bazės procedūra yra skirta korektiškam duomenų įvedimui į pateiktą duomenų bazės struktūrą. Procedūra realizuota naudojant plpgsql kalbą.

```
CREATE OR REPLACE FUNCTION "fnk_add_event" (IN prm_date timestamp, IN prm_sesion_id int4, IN prm_pc_name varchar, IN prm_proc_name varchar, IN prm_pid int4, IN prm_operation varchar, IN prm_path text, IN prm_size int8)
RETURNS int4 AS
$BODY$
        DECLARE _pro_id integer;
        DECLARE _pc_id integer;
        DECLARE _op_id integer;

        BEGIN

        --pc
        SELECT INTO _pc_id pc_id from pc where pc_name ilike prm_pc_name;
        IF _pc_id is null THEN
        INSERT INTO pc (pc_name) VALUES (prm_pc_name);
        _pc_id = currval('pc_pc_id_seq'::regclass);
        END IF;

        --Proccess
        SELECT INTO _pro_id pro_id from proccess where pro_name ilike prm_proc_name AND pro_pc_id = _pc_id AND pro_sesion_id = prm_sesion_id;
        IF _pro_id is null THEN
        INSERT INTO proccess (pro_name, pro_pc_id, pro_sesion_id) VALUES (prm_proc_name, _pc_id, prm_sesion_id);
        _pro_id = currval('proccess_pro_id_seq'::regclass);
```

```
        END IF;


        --Operations
        SELECT INTO _op_id ope_id from operation where ope_name ilike
prm_operation;
        IF _op_id is null THEN
        INSERT INTO operation (ope_name) VALUES (prm_operation);
        _op_id = currval('operation_ope_id_seq'::regclass);
        END IF;
INSERT INTO event (eve_date, eve_pro_id, eve_pid, eve_op_id, eve_path,
eve_size) VALUES
        (prm_date, _pro_id, prm_pid, _op_id, prm_path, prm_size);

        return currval('event_eve_id_seq'::regclass);

            END;
$BODY$
        LANGUAGE plpgsql
        CALLED ON NULL INPUT
        VOLATILE
        EXTERNAL SECURITY INVOKER
        COST 100;
```

Toliau pateikiame vieno vartotojo veiksmų analizę. Buvo analizuojama visa vieno vartotojo veiksmų imtis, kurioje užfiksuota 1020411veiksmai. Duomenys buvo surinkti vartotojo kompiuteryje, kuris buvo imituojamas naudojant jau minėtą virtualią laboratoriją, juos fiksuojant apie 20 valandų, kai kompiuteris buvo įjungtas ir juo naudojosi vartotojas

Ekspertinės sistemos taisyklės buvo paverstos SQL sakiniais, kuriais buvo eliminuojamas tam tikras užklausų skaičius. Pateikiame kelių pavyzdinių ekspertinės sistemos taisyklių SQL sakinius.

Standartinių bylų duomenų bazės patikra:

```
SELECT count(*) FROM event
 JOIN operation on ( eve_op_id = ope_id)
 LEFT JOIN std_file_db on (eve_path = sfd_path)
 where sfd_id is null;
```

Bylų plėtinių duomenų bazės patikra:

```
SELECT count(*) FROM event
 JOIN extensions on (eve_path ilike '%' || ext_name);
```

Vidinės bylų duomenų bazės patikra:

```
SELECT eve_date::date, eve_path, count(*) as kiek FROM event
group by eve_date::date, eve_path
order by kiek desc
```

3.8 lentelė. Ekspertinės sistemos taisyklių eksperimentinių tyrimų rezultatų suvestinė
Table 3.8. Experimental results of rule based expert system

Nr.	Taisyklės pavadinimas	Bendras įvykių sk.	Tikrinimui atrinktas įvykių sk.	Įvykių sumažėjimas proc.
1	Standartinė bylų duomenų bazė	1020411	524381	48,61
2	Bylų plėtinių duomenų bazė	1020411	562912	44,83
3	Vidinė bylų duomenų bazė	1020411	139580	86,32
4	Pagal bylų dydį	1020411	896858	12,11
	Visos taisyklės kartu	1020411	35788	96,49

Pritaikius visas taisykles vieną paskui kitą (eilės tvarka taip pat nėra svarbu), likęs operacijų skaičius yra 35788 vnt. Tai yra 96,49 % sumažėjimas.

Iš 3.8 lentelės duomenų matyti, kad didžiausia įtaką skanuojamų veiksmų skaičiaus sumažinimui turėjo taisyklė nr. 3 „Vidinė bylų duomenų bazė“, kuri sumažino skanuojamą skaičių 86,32 procento. Tai reiškia, kad labiausiai antivirusinės sistemos efektyvumui atsiliepia vis pasikartojantis tų pačių bylų skanavimas. Tuo pačių iš rezultatų matome, kad pakartotinis skanavimas neturi prasmės, kadangi ekspertinės sistemos taikymas nesumažino antivirusinės sistemos taikymo efektyvumo. Blogiausius rezultatus skanuojamų bylų skaičiaus mažinimo prasme pademonstravusi taisyklė nr. 4 “Pagal bylų dydį“ iš esmės atspindi bendrą ypatingai didelių failų skaičių sistemoje, su kuriais susiduria vartotojas. Taisyklės taikymo galimybės ribotos, tačiau atsižvelgiant į taisyklės nufiltruotų bylų dydžių sumą lyginant su visų kitų failų dydžių suma (arti 70 procentų) galime teigti, kad nepaisant procentiškai mažo nufiltruoto bylų skaičiaus, ši taisyklė yra būtina ir kritiškai svarbi antivirusinės sistemos sunaudojamų resursų požiūriu. Taisyklių nr. 1 ir 2 taikymo efektyvumą dar galima didinti, plečiant taisyklių duomenų bazės įrašų skaičių. Atsižvelgiant į tai, kad dvi taisyklės iš keturių dar turi optimizavimo galimybių, galima teigti, kad tobulinant taisykles galutinis sistemos taikymo efektyvumas gali priartėti prie procentinių rodiklių leidžiančių sistemą taikyti praktiškai.

3.5. Trečiojo skyriaus išvados

Skyriuje aptarta naudotos Tele-lab virtualios laboratorijos architektūra, aprašyti atlikti papildomi pritaikymo darbai. Laboratorija buvo išplėsta iki tinklinės, tinkamos kenksmingo programinio kodo tyrimams. Taip pat pasirūpinta jos saugumo klausimais, dirbant su KPK.

Aprašyta ir detalizuota ekspertinė sistema, kuri yra viena svarbiausių kuriamos sistemos dalių. Detaliai aprašytos visos ekspertinės sistemos taisyklės ir jų veikimas. Pateiktas išsamus sukurtos realiojo laiko skenavimo antivirusinės sistemos prototipo aprašymas. Detaliai pateiktos ir pagrįstos prototipo dalys ir metodikos, kurios buvo atrinktos ir panaudotos kūrimo princese.

Eksperimentinių tyrimų metu nustatytas ekspertinės sistemos veikimo našumas, leidęs sumažinti skenuoti siunčiamų užklausų kiekį iki 96 %.

Bendrosios išvados

1. Remiantis eksperimentų rezultatais, buvo parodytas siūlomos ekspertinės sistemos, skirtos atvirojo kodo realiojo laiko antivirusinių sistemų charakteristikoms gerinti, korektiškumas. Sistema gali būti rekomenduota integruoti su atvirojo kodo antivirusinėmis sistemomis. Atlikti bandymai parodė, kad, panaudojus tokio tipo sistemas, atvirojo kodo, parašais pagrįstas antivirusines skenavimo sistemas galima naudoti kompiuterio apsaugai nuo kenksmingo programinio kodo realiuoju laiku.

2. Patvirtinta principinė galimybė saugoti ir interpretuoti ekspertinės sistemos taisykles reliacinės duomenų bazių valdymo sistemos pagrindu, kai yra tiriami dideli veiksmų kiekiai salyginai nedidelėse ekspertinėse sistemose. Pasiūlytas sprendimas gali rasti pritaikymą interneto technologijose dėl plataus RDBVS taikymo masto ir žemo internetinių technologijų integracijos lygio su specializuotomis ekspertinių taisyklių saugojimo ir apdorojimo sistemomis.

3. Ekspertinės sistemos pritaikymas leido sumažinti apkrovą 96,49 % ir nepaisant to nepastebėtas nei vienas virusas iš kenksmingo programinio kodo bazės. Ekspertinės sistemos taisyklių tolimesnis optimizavimas yra įmanomas ir leistų padidinti sistemos efektyvumą iki praktiniam taikymui priimtino lygio.

4. Patvirtinta, kad virtualizuota aplinka gali visiškai emuliuoti visas situacijas susijusias su kenksmingo programinio kodo plitimu. Ši aplinka buvo adaptuota tinklinio kenksmingo programinio kodo atakoms modeliuoti.

Literatūros sąrašas

Additional Information for Windows Vista x64 and Windows 7 x64 Users Only. 2010 [interaktyvus], [žiūrėta 2012 m. sausio 7 d.]. Prieiga per internetą: <http://www.freeotfe.org/docs/Main/impact_of_kernel_driver_signing.htm>.

Bayer, U.; Milani Comparetti, P.; Hlauscheck, C.; Kruegel, C.; Kirda, E. 2009. Scalable, Behavior-Based Malware Clustering, in *16th Symposium on Network and Distributed System Security (NDSS'09).*

Balzarotti, D.; Cova, M.; Karlberger, C.; Kruegel, C.; Kirda, E.; Vigna, G. 2010. Efficient Detection of Split Personalities in Malware, *in Proceedings of the 17th Annual Network and Distributed System Security Symposium (NDSS'10).*

Bellard, F. 2011. QEMU – Open Source Processor Emulater homepage. [online]. Available from Internet: <http://www.qemu.org/>.

Bratus, S.; Locasto, M. E.; Schulte, B. R. 2010. SegSlice: Towards a New Class of Secure Programming Primitives for Trustworthy Platforms, in *The 3rd International Conference on Trust and Trustworthy Computing (TRUST'10)*.

Burghardt, K. 2011. *ClamFS Project Page. Welcome!* [interaktyvus], [žiūrėta 2012 m. balandžio 7 d.]. Prieiga per internetą: <http://clamfs.sourceforge.net>.

Carbone, M.; Cui, W.; Lu, L.; Lee, W., Peinado, M.; Jiang, X. 2009. Mapping Kernel Objects to Enable Systematic Integrity Checking, in *Proceedings of the 16th ACM Conference on Computer and Communications Security (CCS'09).*

Christodorescu, M.; Jha, S. 2003. Static Analysis of Executables to Detect Malicious Patterns, in *Proceedings of the 12th USENIX Security Symposium (Security'03).* Christodorescu, M.; Kruegel, C.; Jha, S. 2007. Mining Specifications of Malicious Behavior, in *Proceedings of the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE'07).*

Clam Sentinel. 2009 [interaktyvus], [žiūrėta 2012 m. vasario 11 d.]. Prieiga per internetą: <http://sourceforge.net/projects/clamsentinel>.

Collberg, C.; Thomborson, C.; Low, D. 1998. Manufacturing Cheap, Resilient, and Stealthy Opaque Constructs, in *Principles of Programming Languages 1998 (POPL'98).*

Čeponis, D.; Radvilavičius, L. 2008. *Windows* API funkcijų stebėjimas, procesų perėmimas ir stebėjimas, iš *11-osios Lietuvos jaunųjų mokslininkų konferencijos „Mokslas – Lietuvos ateitis", įvykusios Vilniuje 2008 m. balandžio 9–11 d., pranešimų medžiaga.* Vilnius: Technika, 391–396.

Čeponis, D.; Radvilavičius, L. 2011. *Windows* API funkcijų sekų perėmimo bibliotekų tyrimas, iš *13-osios Lietuvos jaunųjų mokslininkų konferencijos „Mokslas – Lietuvos ateitis", įvykusios Vilniuje 2010 m. balandžio 16 d., pranešimų medžiaga.* Vilnius: Technika, 15–19.

Čeponis, D.; Radvilavičius, L. Windows API funkcijų perėmimo bibliotekų tyrimas, iš *13-osios Lietuvos jaunųjų mokslininkų konferencijos „Mokslas – Lietuvos ateitis", įvykusios Vilniuje 2010 m. balandžio mėn. 16 d., pranešimų medžiaga.* Vilnius: Technika.

Davis, R. 1992. *Using an Expert System to Peel the Computer Virus Onion*. EDPACS 20(2). 1–12.

Davoli, R. 2011. Virtual Distributed Ethernet homepage [Online]. Available from Internet: <http://vde.sourceforge.com/>.

DeviceTree. 2011 [interaktyvus], [žiūrėta 2012 m. vasario 8 d.]. Prieiga per internetą: <http://www.osronline.com/article.cfm?article=97>.

Dolan-Gavitt, B.; Srivastava, A.; Traynor, P.; Giffin, J. 2009. Robust Signatures for Kernel Data Structures, in *Proceedings of the 16th ACM conference on Computer and communications security (CCS'09)*.

Doser, J. 2003. *Analysis of SecureUML Models*. Freiburg.

Hayes, B. 2010. *Who Goes There?An Introduction to On-Access Virus Scanning, Part One* [interaktyvus], [žiūrėta 2012 m. sausio 5 d.]. Prieiga per internetą: <http://www.symantec.com/connect/articles/who-goes-there-introduction-access-virus-scanning-part-one>.

Handling IRPs: What Every Driver Writer Needs to Know. 2006 [interaktyvus], [žiūrėta 2012 m. vasario 7 d.]. Prieiga per internetą: <http://msdn.*Microsoft*.com/en-us/*Windows*/hardware/gg487398>.

Helenius, M. 2002. *Asystem to support the analysis of Antivirus Products' Virus detection Capabilities*. Academic dissertation. Tempere. 103.

Http://www.clamav.net

http://www.wildlist.org/WildList/ (6.8.2001).

Hu, J.; Cordel, D.; Meinel, C. 2006. A *Virtual Machine Architecture for Creating IT-Security Laboratories*, Technical report. Hasso-Plattner-Insitut.

Hu, J.; Meinel, C. 2004. Tele-Lab IT-Security on CD: Portable, reliable and safe IT security training, *Computers & Security* 23: 282–289.

Hu, J.; Schmitt, M.; Willems, C.; Meinel, C. 2003. A tutoring system for IT-Security, in *Proceedings of the 3rd World Conference in Information Security Education*. Monterey, USA, p. 51–60.

Hund, R.; Holz, T.; Freiling, F. C. 2009. Return-Oriented Rootkits: Bypassing Kernel Code Integrity Protection Mechanisms, in *Proceedings for the 18th USENIX Security Symposium (Security'09).*

Interactive Services [interaktyvus] 2010, [žiūrėta 2010 m. gegužės 29 d.]. Prieiga per internetą: <http://msdn.microsoft.com/en-us/library/ms683502>.

Jacob, G.; Debar, H.; Filiol, E. 2007. Behavioral detection of malware: from a survey towards an established taxonomy. *Journal in Computer Virology* 4(3). 251–266.

Jürjens, J. 2004. *Secure Systems Development with UML*. Springer.

Kernel-Mode Code Signing Policy (Windows Vista and Later). 2012 [interaktyvus], [žiūrėta 2012 m. kovo 20 d.] Prieiga per internetą: <http://msdn.*Microsoft*.com/en-us/library/*Windows*/hardware/ff548231(v=vs.85).aspx>.

Le Charlier B.; Abdelaziz M.; Swimmer M.; Informatik F. 1995. *Dynamic Detection and Classification of Computer Viruses Using General Behaviour Patterns,* 1–22.

Lodderstedt, T.; Basin, D.; Doser, J. 2006. Model Driven Security: from UML Models to Access Control Infrastructure, *ACM Transactions on Software Engineering and Methodology (TOSEM)*, Volume 15, Issue 1.

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

Marti, J. 2011. *noVNC project website* [online]. Available from Internet: <http://kanaka.github.com/noVNC/>.

Method and system for antimalware scanning with variable scan settings. 2010. United States Patent 7725941.

Method and system for delayed write scanning for detecting computer malwares. 2003. United States Patent 7757361.

Method and system for preemptive scanning of computer files. 2010. United States Patent 0242109.

Method and system for antimalware scanning with variable scan setting. 2010. United States Patent 7725941.

Miretskiy, Y.; Das, A.; Wright, C. P., Zadok, E. 2004. Avfs: An On-Access Anti-Virus File System, in *Proceedings of the 13th USENIX Security Symposium, San Diego.*

On-access scan of memory for malware. 2010. United States Patent 0200863.

Okafor, E. C.; Osuagwu, C. C. 2007. Issues in Structuring the Knowledge-base of Expert Systems. *The Electronic Journal of Knowledge Management* (5)3. 313–322

Opferman, T. 2005. *Driver Development Part 1: Introduction to Drivers* [interaktyvus], [žiūrėta 2012 m. kovo 19 d.]. Prieiga per internetą: <http://www.codeproject.com/Articles/9504/Driver-Development-Part-1-Introduction-to-Drivers>.

Phrack Magazine. Linux on-the-fly kernel patching without LKM. Prieiga per internetą: <http://www.phrack.com/issues.html?issue=58&id=7>.

Ray, I.; Li, N.; France, R.; Kim, D.-K. 2004. Using UML To Visualize Role-Based Access Control Constraints, in *Proceedings of the ninth ACM symposium on Access control models and technologies*, 115–124, Yorktown Heights, New York, USA.

Red Hat, Inc. 2011. Kernel-based Virtual Machine (KVM) homepage [online]. Available from Internet: <http://www.linux-kvm.org/>.

Riley, R.; Jiang, X.; Xu, D. 2008. Guest-Transparent Prevention of Kernel Rootkits with VMM-based Memory Shadowing, in *Proceedings of 11th International Symposium on Recent Advances in Intrusion Detection (RAID'08)*.

Riley, R.; Jiang, X.; Xu, D. 2009. Multi-Aspect Profiling of Kernel Rootkit Behavior, in *Proceedings of the 4th European Conference on Computer Systems (Eurosys'09).*

Ruili Zhou, Jianfeng Pan, Xiaobin Tan, Hongsheng Xi, *Application of CLIPS Expert System to Malware Detection System*, cis, vol. 1, pp. 309–314, 2008 International Conference on Computational Intelligence and Security, 2008

Schneier, B. 1996. *Applied Cryptography.* John Wiley & Sons.

Scroggins, R. 2009. *The Clam Sentinel Program Description And Setup* [interaktyvus], [žiūrėta 2012 m. vasario 11 d.]. Prieiga per internetą: <http://clamsentinel.sourceforge.net/SentinelSimpleGuide.html>.

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>.

Seshadri, A.; Luk, M.; Qu, N.; Perrig, A. 2007. SecVisor: A Tiny Hypervisor to Provide Lifetime Kernel Code Integrity for Commodity OSes, in *Proceedings of 21st Symposium on Operating Systems Principles (SOSP'07). ACM.*

Sharif, M.; Lanzi, A.; Giffin, J.; Lee, W. 2009. Impeding Malware Analysis Using Conditional Code Obfuscation, in *Proceedings of the 15th Annual Network and Distributed System Security Symposium (NDSS'08).*

System and method of caching decisions on when to scan for malware. 2006. United States Patent 7882561.

Szor, P. 2005. *The Art of Computer Virus Research and Defense*. Hagerstown: Addison-Wesley Professional.

The Libvirt Developers. 2011 libvirt – The virtualization API homepage. [online]. Available from Internet: <http://libvirt.org/>

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

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

Trendy, C. 1996 *Wacky widgets. wacky cost: false positives*. Virus Bulletin Journal May 1996, Virus Bulletin Ltd., 21 The Quadrant, Abingdon, Oxfordshire, OX14 3YS, England pp. 16–17.

Unload routine. 2012 [interaktyvus], [žiūrėta 2012 m. kovo 19 d.]. Prieiga per internetą: <http://msdn.*Microsoft*.com/en-us/library/*Windows*/hardware/ff564886(v=vs.85).aspx>.

Virtual file system. 2012 [interaktyvus], [žiūrėta 2012 m. balandžio 5 d.]. Prieiga per internetą: <http://en.wikipedia.org/wiki/Virtual_file_system>.

Wang, C.; Hill, J.; Knight, J. C.; Davidson, W. J. 2001. Protection of Software-Based Survivability Mechanisms, in *Proceedings of the 2001 International Conference on Dependable Systems and Networks (DSN'01)*.

Wang, L.; Tan, X. ; Pan, J.; Xi, H. 2009. Application of PrefixSpan* Algorithm in Malware Detection Expert System. *Education Technology and Computer Science, 2009. ETCS '09. First International Workshop on*. (3) 448– 452.

Willems, C.; Meinel, C. 2008. Tele-Lab IT-Security: an Architecture for an online virtual IT Security Lab, *International Journal of Online Engineering* (iJOE), X.

Windows Driver Kit Version 7.1.0. 2010 [interaktyvus], [žiūrėta 2012 m. vasario 5 d.]. Prieiga per internetą: <http://www.*Microsoft*.com/download/en/details.aspx?id=11800>.

Xuan, C.; Copeland, J. A.; Beyah, R. A. 2009. Toward Revealing Kernel Malware Behavior in Virtual Execution Environments, in *Proceedings of 12th International Symposium on Recent Advances in Intrusion Detection (RAID'09).*

Autoriaus publikacijų disertacijos tema sąrašas

Straipsniai recenzuojamuose mokslo žurnaluose

Čenys, A.; Normantas, A.; Radvilavičius, L. 2009. Designing role-based access control policies with UML, *Journal of Engineering Science and Technology Review* 2(1): 48–50. ISSN 1791-2377.

Radvilavičius, L.; Čeponis, D. 2011. Window API funkcijų sekų perėmimo bibliotekų tyrimas, *Mokslas – Lietuvos ateitis* 3(1): 15–19. ISSN 2029-2341.

Radvilavičius, L.; Marozas, L.; Čenys, A. 2012. Overview of real-time antivirus scanning engines, *Journal of Engineering Science and Technology Review* 5(1): 63–71. ISSN 1791-2377.

Straipsniai kituose leidiniuose

Willems, Ch.; Klingbeil, Th.; Radvilavičius, L.; Čenys, A. 2011. A distributed virtual laboratory architecture for cybersecurity training, in *Proceedings 6th International Conference on Internet Technology and Secured Transactions (ICITST-2011)*, 11–14 December 2011, Abu Dhabi, United Arab Emirates. Abu Dhabi: IEEE, p. 408–415. ISBN 9781908320001. Prieiga per internetą: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6148474>.

Radvilavičius, L.; Talmontienė, J. 2012. Antivirusinių programų failinės sistemos stebėjimo realiu laiku metodų tyrimas, *Jaunųjų mokslininkų darbai* 3(36): 116–122.

Priedas

Duomenų bazės architektūra ir užklausų sakiniai

Duomenų bazės architektūra

Duomenų lentelė „Event“

Aprašymas. Duomenų lentelėje saugomi agreguoti duomenys apie naudotojų kompiuteriuose atliktus veiksmus.

Duomenų bazės lentelės sukūrimo sakinys:

```
CREATE TABLE "event" (
        "eve_id" BIGSERIAL NOT NULL,
        "eve_date" timestamp(0),
        "eve_pro_id" int4 NOT NULL,
        "eve_pid" int4,
        "eve_op_id" int4,
        "eve_path" text,
        "eve_size" int8,
        "eve_file_size" int8,
 CONSTRAINT "event_pkey" PRIMARY KEY("eve_id"),
 CONSTRAINT "fk_ope_eve" FOREIGN KEY ("eve_op_id")
  REFERENCES "operation"("ope_id")
```

```
        MATCH SIMPLE
        ON DELETE CASCADE
        ON UPDATE CASCADE
        NOT DEFERRABLE,
 CONSTRAINT "fk_pro_eve" FOREIGN KEY ("eve_pro_id")
 REFERENCES "proccess"("pro_id")
        MATCH SIMPLE
        ON DELETE CASCADE
        ON UPDATE CASCADE
        NOT DEFERRABLE
)
WITHOUT OIDS;
CREATE INDEX "fki_ope_eve" ON "event" USING BTREE (
        "eve_op_id"
);
CREATE INDEX "fki_pro_eve" ON "event" USING BTREE (
        "eve_pro_id"
);
```

1 lentelė. Duomenų lentelė „Event“
Table1. Database table „Event“

Column Name	Data Type	Primary Key	Not Null	AutoInc	Komentarai
eve_id	int8	YES	YES	YES	
eve_date	timestamp(0)	NO	NO	NO	
eve_pro_id	int4	NO	YES	NO	
eve_pid	int4	NO	NO	NO	
eve_op_id	int4	NO	NO	NO	
eve_path	text	NO	NO	NO	
eve_size	int8	NO	NO	NO	
eve_file_size	int8	NO	NO	NO	in bytes

Duomenų lentelė „Extensions“
Aprašymas: duomenų lentelėje saugomi bylų plėtinių pavadinimai.
Duomenų bazės lentelės sukūrimo sakinys:

```
CREATE TABLE "extensions" (
        "ext_id" SERIAL NOT NULL,
        "ext_name" varchar(5),
 CONSTRAINT "extensions_pkey" PRIMARY KEY("ext_id")
)
WITHOUT OIDS;
```

```
CREATE UNIQUE INDEX "ext_name_idx" ON "extensions" USING BTREE (
        "ext_name"
);
```

2 lentelė. Duomenų lentelė „Extensions“
Table 2. Database table „Extensions“

Column Name	Data Type	Primary Key	Not Null	AutoInc	Komentarai
ext_id	int4	YES	YES	YES	
ext_name	varchar(5)	NO	NO	NO	

Duomenų lentelė „Operation“
Aprašymas: duomenų lentelėje saugomi operacijų su bylomis pavadinimai.
Duomenų bazės lentelės sukūrimo sakinys:

```
CREATE TABLE "operation" (
        "ope_id" SERIAL NOT NULL,
        "ope_name" varchar,
 CONSTRAINT "operation_pkey" PRIMARY KEY("ope_id")
)
WITHOUT OIDS;
CREATE UNIQUE INDEX "ope_id_index" ON "operation" USING BTREE (
        "ope_id"
);
CREATE UNIQUE INDEX "ope_name_index" ON "operation" USING
BTREE (
        "ope_name"
);
```

3 lentelė. Duomenų lentelė „Operation“
Table 3. Database table „Operation“

Column Name	Data Type	Primary Key	Not Null	AutoInc	Komentarai
ope_id	int4	YES	YES	YES	
ope_name	varchar	NO	NO	NO	

Duomenų lentelė „Pc“
Aprašymas: Duomenų lentelėje saugomi vartotojų kompiuterių pavadinimai.
Duomenų bazės lentelės sukūrimo sakinys:

```
CREATE TABLE "pc" (
        "pc_id" SERIAL NOT NULL,
        "pc_name" varchar,
```

```
 CONSTRAINT "pc_pkey" PRIMARY KEY("pc_id")
)
WITHOUT OIDS;
```

4 lentelė. Duomenų lentelė „Pc“
Table 4. Database table „Pc“

Column Name	Data Type	Primary Key	Not Null	AutoInc	Komentarai
pc_id	int4	YES	YES	YES	
pc_name	varchar	NO	NO	NO	

Duomenų lentelė „Proccess“
Aprašymas: duomenų lentelėje saugomi procesų pavadinimai ir sesijų identifikatoriai.
Duomenų bazės lentelės sukūrimo sakinys:

```
CREATE TABLE "proccess" (
        "pro_id" SERIAL NOT NULL,
        "pro_name" varchar(100),
        "pro_pc_id" int4,
        "pro_sesion_id" int4,
 CONSTRAINT "proccess_pkey" PRIMARY KEY("pro_id"),
 CONSTRAINT "unq_pro_name_pc"
UNIQUE("pro_name","pro_pc_id","pro_sesion_id"),
 CONSTRAINT "fk_pc" FOREIGN KEY ("pro_pc_id")
  REFERENCES "pc"("pc_id")
        MATCH SIMPLE
        ON DELETE NO ACTION
        ON UPDATE NO ACTION
        NOT DEFERRABLE
)
WITHOUT OIDS;

CREATE INDEX "fki_pc" ON "proccess" USING BTREE (
        "pro_pc_id"
);
```

5 lentelė. Duomenų lentelė „Proccess“
Table 5. Database table „Proccess“

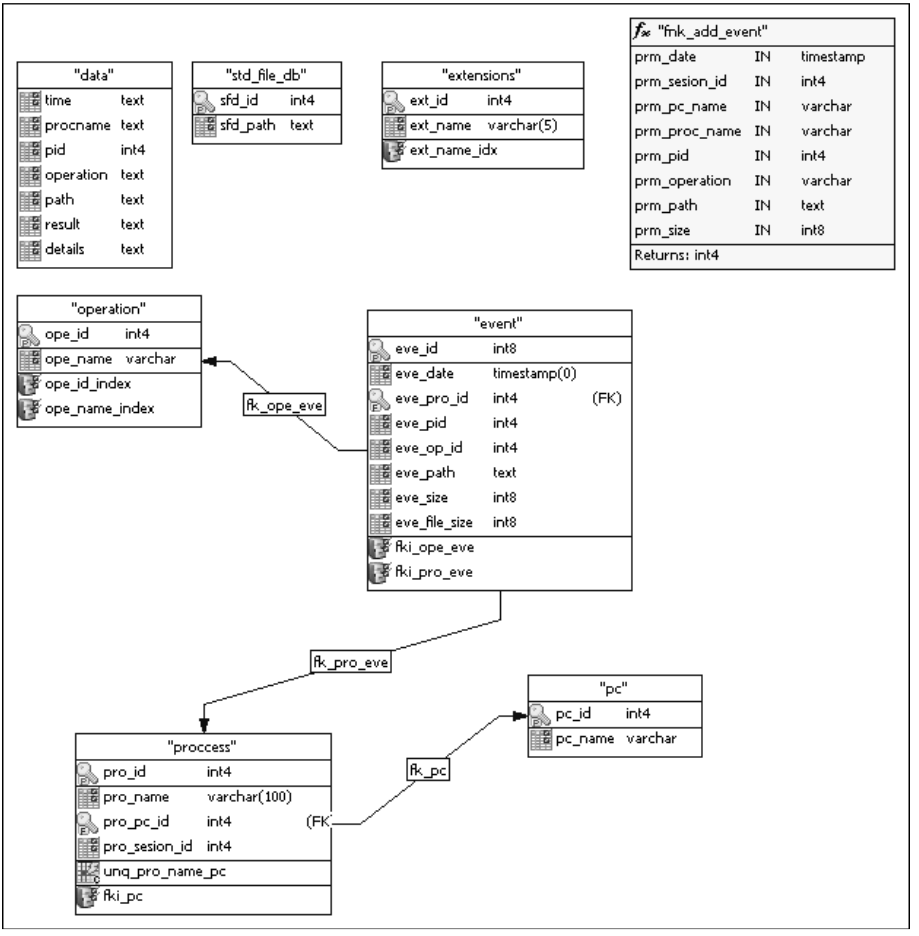
Column Name	Data Type	Primary Key	Not Null	AutoInc	Komentarai
pro_id	int4	YES	YES	YES	
pro_name	varchar(100)	NO	NO	NO	
pro_pc_id	int4	NO	NO	NO	
pro_sesion_id	int4	NO	NO	NO	

Duomenų lentelė „std_file_db“
Aprašymas: duomenų lentelėje saugoma standartinių bylų duomenų bazės informacija.
Duomenų bazės lentelės sukūrimo sakinys:

```
CREATE TABLE "std_file_db" (
        "sfd_id" SERIAL NOT NULL,
        "sfd_path" text,
 CONSTRAINT "std_file_sdb_pkey" PRIMARY KEY("sfd_id")
)
WITHOUT OIDS;
```

6 lentelė. Duomenų lentelė „std_file_db“
Table 6. Database table „std_file_db“

Column Name	Data Type	Primary Key	Not Null	AutoInc	Komentarai
sfd_id	int4	YES	YES	YES	
sfd_path	text	NO	NO	NO	



1 pav. Duomenų bazės schema
Fig. 1. Datebase scheme

Duomenų bazės procedūra yra skirta korektiškam duomenų įvedimui į pateiktą duomenų bazės struktūrą. Procedūra realizuota naudojant plpgsql kalbą.

```
CREATE OR REPLACE FUNCTION "fnk_add_event" (IN prm_date timestamp, IN prm_sesion_id int4, IN prm_pc_name varchar, IN prm_proc_name varchar, IN prm_pid int4, IN prm_operation varchar, IN prm_path text, IN prm_size int8)
RETURNS int4 AS
$BODY$
        DECLARE _pro_id integer;
```

```
	DECLARE _pc_id integer;
	DECLARE _op_id integer;

	BEGIN

	--pc
	SELECT INTO _pc_id pc_id from pc where pc_name ilike
prm_pc_name;
	IF _pc_id is null THEN
	INSERT INTO pc (pc_name) VALUES (prm_pc_name);
	_pc_id = currval('pc_pc_id_seq'::regclass);
	END IF;

	--Proccess
	SELECT INTO _pro_id pro_id from proccess where pro_name ilike
prm_proc_name AND pro_pc_id = _pc_id AND pro_sesion_id =
prm_sesion_id;
	IF _pro_id is null THEN
	INSERT INTO proccess (pro_name, pro_pc_id, pro_sesion_id)
VALUES (prm_proc_name, _pc_id, prm_sesion_id);
	_pro_id = currval('proccess_pro_id_seq'::regclass);
	END IF;


	--Operations
	SELECT INTO _op_id ope_id from operation where ope_name ilike
prm_operation;
	IF _op_id is null THEN
	INSERT INTO operation (ope_name) VALUES (prm_operation);
	_op_id = currval('operation_ope_id_seq'::regclass);
	END IF;
INSERT INTO event (eve_date, eve_pro_id, eve_pid, eve_op_id, eve_path,
eve_size) VALUES
	(prm_date, _pro_id, prm_pid, _op_id, prm_path, prm_size);

	return currval('event_eve_id_seq'::regclass);

	END;
$BODY$
	LANGUAGE plpgsql
	CALLED ON NULL INPUT
```

```
        VOLATILE
        EXTERNAL SECURITY INVOKER
        COST 100;
```

Panaudoti SQL užklausų sakiniai:

```
SELECT count(*) FROM event
 JOIN operation on ( eve_op_id = ope_id)
 LEFT JOIN std_file_db on (eve_path = sfd_path)
 where sfd_id is null

SELECT count(*) FROM event
 JOIN extensions on (eve_path ilike '%' || ext_name)

SELECT eve_date::date, eve_path, count(*) as kiek FROM event
 group by eve_date::date, eve_path
 order by kiek desc
```

Lukas RADVILAVIČIUS

REALIOJO LAIKO SKENAVIMO ANTIVIRUSINIŲ SISTEMŲ NAŠUMO CHARAKTERISTIKŲ TYRIMAS

Daktaro disertacija

Technologijos mokslai,
Informatikos inžinerija (07T)

REAL-TIME ANTIVIRUS SCANNING METHODS CHARACTERISTICS' EFFICIENCY STUDY

Doctoral Dissertation

Technological Sciences,
Informatics Engineering (07T)

2012 12 28. 9,5 sp. l. Tiražas 20 egz.
Vilniaus Gedimino technikos universiteto
leidykla „Technika“,
Saulėtekio al. 11, 10223 Vilnius,
http://leidykla.vgtu.lt
Spausdino UAB „Ciklonas“
J. Jasinskio g. 15, 01111 Vilnius