KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
KOMPIUTERIŲ KATEDRA

Darius Rudzika

Piktavališkos programinės įrangos virtualių mašinų aplinkoje aptikimo metodikos sudarymas ir tyrimas

Magistro darbas

Darbo vadovas
doc. dr. J. Toldinas

Kaunas, 2010

KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
KOMPIUTERIŲ KATEDRA

Darius Rudzika

Piktavališkos programinės įrangos virtualių mašinų aplinkoje aptikimo metodikos sudarymas ir tyrimas

Magistro darbas

Recenzentas
doc. dr. R. Damaševičius
2010-05-

Vadovas
doc. dr. J. Toldinas
2010-05-

Atliko
IFN-8/3 gr. stud.
Darius Rudzika
2010-05-15

Kaunas, 2010

Turinys

1. Įvadas

Pagal šiuolaikinės ICT rinkos tendencijas informacinės technologijos globaliai juda link virtualizacija paremtų sprendimų. Darbo vietos perkeliamos iš įprastų asmeninių kompiuterių į virtualias darbo vietas, o vėliau ir į debesų kompiuterijos (angl. cloud computing) sprendimus. Vis dažniau pasirodo virtualizacijos produktai nešiojamiems įrenginiams – delninukams, išmaniesiems telefonams. Didėjant pasaulinėms efektyvaus energijos ir infrastruktūros panaudojimo tendencijoms, informacinių sistemų diegime vis plačiau naudojamos virtualizacijos technologijos.

Virtualizacija leidžia efektyviai spręsti skaičiavimo resursų panaudojimo uždavinius, valdyti duomenų centre suvartojamos elektros energijos kaštus, lanksčiai diegti kelias, skirtingų gamintojų, operacines sistemas vienoje tarnybinėje stotyje, tuo pačiu sumažinant fizinių tarnybinių stočių kiekį, jiems reikalingos elektros ir aptarnavimo kaštus.
Išspręsdama didelę dalį informacinių technologijų problemų virtualizacija, sukuria nemažai naujų iššūkių. Dalis jų – informacijos saugos problemos.

Kaip paaiškėjo šio darbo analizės etape - piktavališkos programinės įrangos grėsmės išlieka aktualios ir virtualizuotose operacinėse sistemose, o tam tikrais atvejais netgi padidėja. Todėl turime imtis priemonių apsaugoti virtualizuotas informacines sistemas nuo piktavališkos programinės įrangos. Piktavališka programinė įranga pagal naudotos literatūros šaltinių klasifikaciją [23] skirstoma į keturis tipus: 0 tipo, I tipo, II ir III tipo. Pagal Microsoft korporacijos gaminamo produkto - Malicious Software Removal Tool statistiką, didelė dalis aptinkamos piktavališkos programinės įrangos yra “rootkit” tipo. Pagal minėtą klasifikaciją “rootkit” tipo piktavališka programinė įranga priklauso I ir II tipui[23]. „Rootkit“ tipo piktavališka programinė įranga yra maža programa, kuri slapta įsibrauna į operacinę sistema ar jos branduolį ir perima kompiuterio ar tarnybinės stoties valdymą [1]. Dabar labiau nei bet kada, “rootkit” tipo piktavališka programinė įranga, susilaukia informacinių sistemų saugos analitikų dėmesio. Pagrindinė to priežastis šio tipo piktavališka programinė įranga dengia daug paslėptos veiklos – paslepia kenkėjiškus operacinės sistemos procesus, suteikia administratoriaus arba sistemos privilegijas ir kitaip išnaudoja operacinės sistemos branduolio pažeidžiamumus.

„Rootkit“ piktavališką programinę įrangą galima suskirstyti į dvi grupes: vartotojo lygmens (angl. User-level, application-level) ir branduolio lygmens (kernel - level) [23].

Vartotojo lygmens “rootkit” programas galima nesunkiai aptikti nes jos nemodifikuoja OS branduolio ir kitų komponentų. Tuo tarpu antroji, branduolio lygmens, grupė gerokai

pavojingesnė, nes administratoriams pateikia suklastotą informaciją apie operacinę sistemą, kurioje veikia – paslepia procesus, sistemos vartotojus ir per tinklą prieinamus resursus.

Šio darbo tikslas ištirti metodus tinkamus klastingos, “rootkit” tipo, piktavališkos programinės įrangos aptikimui virtualių mašinų aplinkose, juos pritaikyti piktavališkos programinės įrangos aptikimo metodikai sudaryti. Metodiką pritaikyti tyrime ir aprašyti rezultatus.

Magistro darbo rezultatai buvo pristatyti tarptautinėje konferencijoje „Elektronika 2010“. Paruoštas publikacijai straipsnis: „Rootkit detection experiment within a virtual environment“. Konferencijos dalyvio diplomas ir straipsnio tekstas pateikti prieduose.

Darbo atsiradimą sąlygojusios priežastys:

- ✓ Virtualizuotos tarnybinės stotys išlaiko tas pačias operacinių sistemų saugumo problemas, kaip ir iki virtualizacijos, tačiau papildomai atsiranda naujos saugos problemos:
- ✓ Virtualizavimo lygmuo (hipervizorius) tai naujo pobūdžio operacinė sistema
- ✓ Virtualizavimo lygmuo (hipervizorius) tai dar vienas piktavalių atakų taikinys
- ✓ Atsiranda atakos galimybė iš vienos s*večio* sistemos prieš kitą *svečio* sistemą, kurios tuo pat metu yra viename fiziniame serveryje.
- ✓ Virtualizacija įgalina platinti programinę įrangą naudojant virtualizuotus (OS + programinė įranga) paketus. Paketų patikimumas turi savus saugos klausimus.

Darbo prielaidos:

Darbe koncentruojamasi į plataus masto problematiką. Viso darbo metu naudojami Intel x86 architektūros pagrindu dirbantys komponentai. Kadangi vyraujanti dauguma pasaulyje pagaminamų kompiuterių yra Intel x86 architektūros, atitinkamai, tiek saugos problemų, tiek virtualizacijos sprendimų šiai platformai yra daugiausiai. Darbe apžvelgiami virtualizacijos metodai ir sprendimai paremti Intel x86, x86-64 ir x64 technologijomis, tačiau jais neapsiriboja. Darbo metu naudojama gerai žinoma piktavališka programinė įranga su tikslu užtikrinti prognozuojamą laboratorinę aplinką. Siekiant užtikrinti eksperimento atkartojamumą darbo metu naudojama tik - nemokama, nemokama akademinei aplinkai, atviro kodo ir laisvai prieinama programinė įranga. Darbo metu gilinamasi tik į „rootkit” tipo klastingos piktavališkos programinės įrangos aptikimo metodus ir jų našumą. Eksperimento metu, „svečio“ naudojama operacinė sistema - Microsoft Windows XP. Darbo metu

naudojama „šeimininko“ operacinė sistema - Windows 7, ant jos dirba hipervizorius ir darbo metu ji laikoma nepažeidžiama.

Metodologija:

Pagrindinė darbe naudojama metodika piktavališkos programinės įrangos aptikimui – virtualios mašinos darbinės atminties analizė. Šiuo metu prieinamiausias metodas atminties pavyzdžiams , kurie buvo naudojami tyrime, rinkti – momentinės virtualių mašinų kopijos.
Tai yra patogu ir efektyvu, nes kartu su virtualios mašinos, būsena išsaugoma ir darbinė atmintis. Kiti galimi metodai pvz. hipervizoriaus funkcionalumas leidžiantis skaityti virtualių mašinų atmintį, bus apžvelgti analizės dalyje, tačiau dėl uždaro kodo arba uždaro licencijavimo apribojimų realizacijoje ir eksperimentuose nebus naudojami.

Tiriamasis objektas ir sritis:

- ✓ Piktavališka programinė įranga
- ✓ Virtuali aplinka, virtualizavimo priemonės ir virtualizuotos operacinės sistemos
- ✓ Piktavališkos programinės įrangos aptikimo metodika
- ✓ Aptikimo metodikos tinkamumas virtualizuotoms aplinkoms

Darbo struktūra:

- ✓ Darbo analizės dalyje apžvelgiami šiuolaikinės virtualizacijos sprendimai ir jų panaudojimas. Naudojantis kitų tyrėjų paskelbtais rezultatais atliekama piktavališkos programinės įrangos veikimo principų apžvalga, klasifikavimo pagal veikimo ypatumus ir aptikimo būdus analizė. Gilinamasi į problemas, kurios atsiranda virtualizavus informacines sistemas. Aprašomi galimi saugos virtualizuotose aplinkose realizavimo būdai ir jų veiksmingumas.
- ✓ Metodikos kūrimo ir eksperimento dalyje parenkami tinkamiausi metodai ir įrankiai piktavališkai programinei įrangai aptikti ir atliekamas metodikos realizavimas. Realizuotais metodais, ištiriami virtualios aplinkos parametrai reikalingi piktavališkos programinės įrangos aptikimui.
- ✓ Eksperimento dalyje surinkti rezultatai pateikiami ir išnagrinėjami detaliau.
- ✓ Pabaigoje pateikiamos atlikto darbo pagrindinės išvados ir rezultatai
- ✓ Prieduose pateikiama papildoma su darbu susijusi informacija – eksperimento rezultatai, mokslinis straipsnis, dalyvavimą konferencijoje pažymintis diplomas ir programinės įrangos rinkinys naudotas eksperimento atlikimo metu.

2. Piktavališkos programinės įrangos virtualiose aplinkose aptikimo metodų analizė

Analizės tikslas – išanalizuoti mokslinius straipsnius apie piktavališkos programinės įrangos aptikimą operacinėse sistemose. Peržvelgti virtualizacijos realizacijas ir jų ypatumus. Ištirti kitų tyrėjų metodus, bei sprendimus, tam, kad būtų galima efektyviai ir sparčiai aptikti piktavališką programinę įrangą.

2.1. Kompiuterių resursų virtualizacija ir virtualių aplinkų panaudojimas

Tradiciniuose duomenų centruose visi fiziniai komponentai tarpusavyje yra tvirtai susiję statiniais ryšiais. Tarnybinės stotys susijusios su joms priskirtais diskų masyvais, tinklo plokštės su atitinkamais prievadais tinklo komutatoriuose, programinė įranga yra tvirtai susijusi su operacine sistema ir tarnybine stotimi, kurioje ji dirba. Virtualizuota infrastruktūra leidžia šiuos griežtus statinius ryšius pakeisti dinaminiais ryšiais.

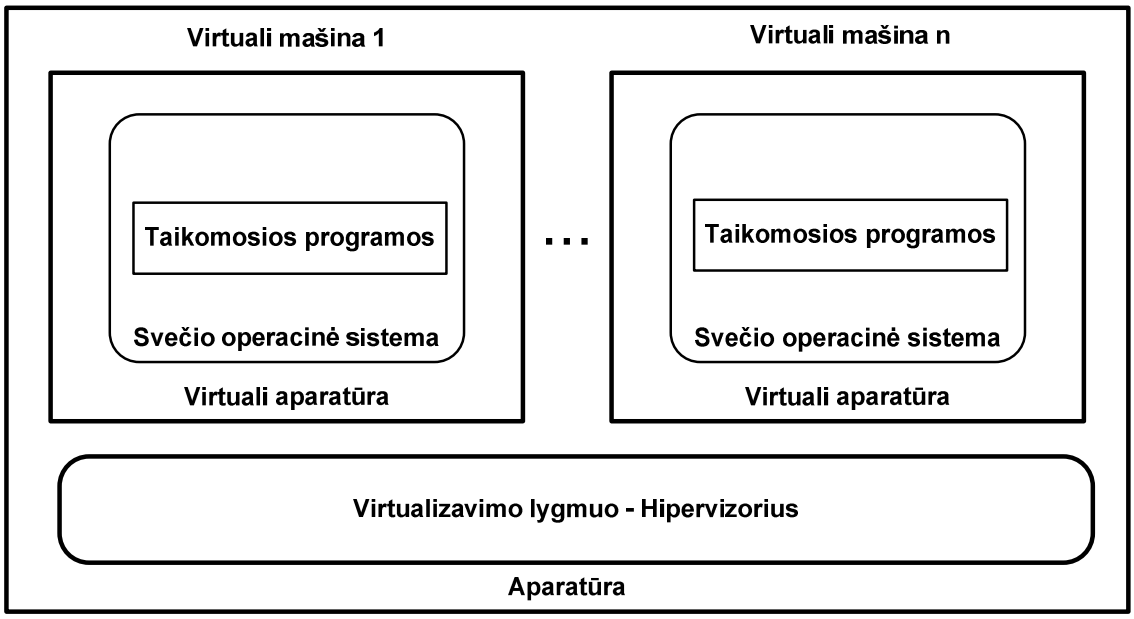
Virtualizacija sukuria tarpinį lygmenį – hipervizorių, atliekantį resursų valdytojo vaidmenį ir atskiriantį operacines sistemas nuo aparatūros, kurioje jos veikia, tuo įgalindama veikti kelias virtualizuotas operacines sistemas (virtualias mašinas) vienoje tarnybinėje stotyje. Pagrindinis virtualizacijos realizavimo komponentas – hipervizorius.

Pagal R. Goldberg [11] yra išskiriamos dvi pagrindinės virtualizacijos architektūros arba du hipervizorių tipai – I tipo ir II tipo.

I tipo (native, bare-metal) hipervizorius. (1 pav.) Šio tipo hipervizorius dirba betarpiškai su tarnybinės stoties aparatine įranga. Virtualizuota – „svečio“ operacinė sistema dirba antrajame virš hipervizoriaus esančiame lygmenyje.

Dažniausiai I tipo hipervizorius yra realizuojamas mikrobranduolio pagrindu, aparatūra ir virtualias mašinas skiriantį lygmenį dažniausiai sudaro - procesoriaus, darbinės atminties ir diskinių kaupiklių resursų valdymo funkcijos.

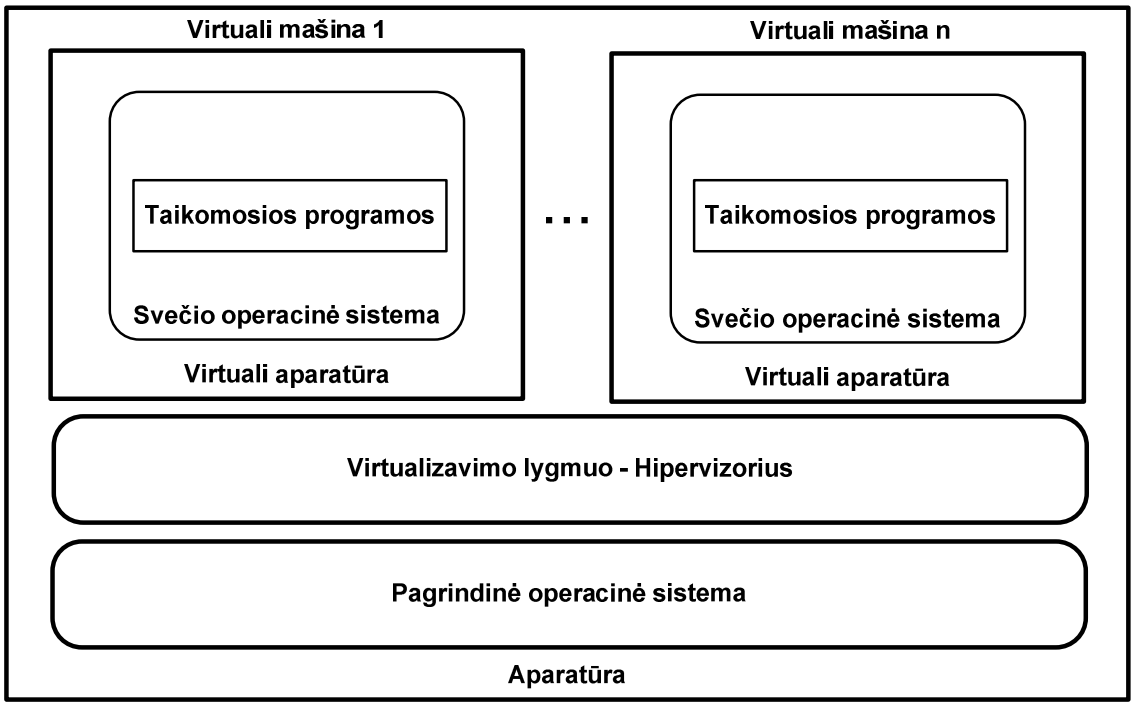
Šis modelis yra klasikinė virtualių mašinų architektūros realizacija. Pirmoji šio tipo hipervizoriaus realizacija buvo 1960 metais IBM sukurtas CP/CMS hipervizorius, IBM z/VM pirmtakas. Pagal šį modelį realizuoti šiuo metu populiariausi hipervizoriai – VMware ESX, Microsoft Hyper-V, Citrix Xen Server.



1 pav. – I tipo hipervizorius

II tipo (hosted) hipervizorius (2 pav.) – dirba įprastinėje pilnavertėje operacinėje sistemoje (dažniausiai Windows arba Linux). Tokiu būdu virtualizuota – „svečio“ operacinė sistema dirba trečiame lygmenyje.

Dažniausiai ši virtualizacijos architektūra naudojama darbo stočių užduotims virtualizuoti. Jei tokia virtualizacija naudojama darbo stotyje, lygiagrečiai virtualioms mašinoms gali būti leidžiamos pagrindinės operacinės sistemos taikomosios programos.



2 pav. II tipo hipervizorius

Detalesnis I ir II tipo hipervizorių savybių palyginimas pateiktas Lentelėse 1 ir 2.

Lentelė 1. I tipo populiariausių hipervizorių apžvalga

Pavadinimas	**Kūrėjas**	**Pagrindinis CPU**	**Svečio CPU**	**Svečio OS**	**Licencija**
VMware ESX	VMware	x86; x86-64	x86; x86-64	Windows, Linux, Solaris, FreeBSD, Virtual appliances, Netware, OS/2, SCO, BeOS, Darwin	Uždara
VMware ESXi	VMware	x86; x86-64	x86; x86-64	Windows, Linux, Solaris, FreeBSD, Virtual appliances, Netware, OS/2, SCO, BeOS, Darwin	Uždara
Hyper-V	Microsoft	x64 +aparatinis palaikymas Intel VT-x arba AMD-V	x64; x86	Palaikomos tvarkyklės Windows 2000, Windows 2003, Windows 2008, Windows XP, Windows Vista, Linux	Uždara
Citrix XenServer	Citrix Systems	x86, x86-64 and IA-64	x86, x86-64 and IA-64	FreeBSD, NetBSD, Linux, Solaris, Windows XP ir 2003 Server (reikalauja vers. 3.0 ir Intel VT-x (Vanderpool) arba AMD-V (Pacifica) CPU, Plan 9	GPL

Šiuo metu virtualizacijos technologijų lyderis yra kompanija VMware, tačiau kiti gamintojai, tokie kaip Microsoft ir Citrix, neatsilieka ir siūlo aibę konkurencingų virtualizacijos sprendimų.

Lentelė 2. II tipo populiariausių hipervizorių apžvalga

Pavadinimas	**Kūrėjas**	**Pagrindinis CPU**	**Svečio CPU**	**Svečio OS**	**Licencija**
VMware Workstation	VMware	x86, x86-64	x86, x86-64	Windows, Linux, Solaris, FreeBSD, Virtual appliances, Netware, OS/2, SCO, BeOS, Darwin	Uždara
VMware Server	VMware	x86, x86-64	x86, x86-64	Windows, Linux, Solaris, FreeBSD, Virtual appliances, Netware, OS/2, SCO, BeOS, Darwin	Uždara
VMware Fusion	VMware	x86, x86-64	x86, x86-64	Windows, Linux, Solaris, FreeBSD, Virtual appliances, Netware, OS/2, SCO, BeOS, Darwin	Uždara
Microsoft Virtual PC 2007	Microsoft/ Connectix	x86, x86-64	x86	DOS, Windows, OS/2, Linux(Suse, Xubuntu), OpenSolaris (Belenix)	Uždara
Windows Virtual PC	Microsoft/ Connectix	x86, x86-64	x86	Windows XP SP3, Windows Vista, Windows 7	Uždara
Microsoft Virtual Server 2005 R2	Microsoft	Intel x86, AMD64	Intel x86	Windows NT, 2000, 2003, Linux (Red Hat and SUSE)	Uždara
Parallels Workstation	Parallels Inc.	x86, Intel VT-x	x86	Windows, Linux, FreeBSD, OS/2, eComStation, MS-DOS, Solaris	Uždara

2.2. Rizikų įvertinimas

Kai kalba eina apie naujos technologijos priėmimą į eksploataciją daugelis organizacijų turi griežtus ir struktūrizuotus procesus, kurie įvertina daugelį saugos aspektų.

Tačiau tarnybinių stočių virtualizacija į duomenų centrus atkeliavo vedama kitų paskatų, nei visos kitos technologijos.
Pirmiausiai ji buvo pozicionuojama kaip tarnybinių stočių konsolidacijos technologija leidžianti mažinti elektros energijos ir eksploatacijos kaštus, kas leido jai apeiti įprastinius saugos vertinimus.

Šiandien virtualizacijos vertė duomenų centre yra neginčijama. Todėl dabartinėje stadijoje priversti virtualizaciją praeiti visas tradicines saugos įvertinimo patikras nepavyks, belieka įvertinti susidariusią situaciją ir pasirinkti tinkamas rizikos valdymo priemones.

Atsiradus virtualizacijos galimybei daug programinės įrangos gamintojų pradėjo savo produktus paketuoti į virtualias mašinas kartu su jų programinei įrangai optimizuotomis operacinėmis sistemomis. Toks metodas leidžia užtikrinti geresnę programinio produkto pateikimo kokybę, tačiau, žiūrint iš informacijos saugos pozicijų, matome daug neigiamų aspektų:

- ✓ Jei operacinė sistema pateikiama specialiai apdorota, jai nebegalioja įprastiniai atnaujinimo scenarijai, nes jie gali sukelti nepageidaujamas pasekmes pakete esančiai programinei įrangai. Tačiau, jei atnaujinimai nebus savalaikiai, kils pavojus pačiai operacinei sistemai ir infrastuktūrai, kurioje ji eksploatuojama.
- ✓ Jei operacinė sistema atnaujinama, nuotoliniu būdu, programinės įrangos gamintojo, reikia įvertinti jo saugų prisijungimą į duomenų centrą.

Prieš naudojant programinę įrangą platinamą virtualių mašinų paketais, reiktų įvertinti šias rizikas:

- ✓ Kiek yra apsaugota (sumažinta, padidinto atsparumo, neprieinama) programinės įrangos gamintojo pateikiama OS?
- ✓ Ar yra apribota galimybė įkelti ir aktyvuoti papildomą programinį kodą?
- ✓ Ar nėra potencialių problemų tarp VM paketo ir tinklo įrangos arba kitų infrastruktūros komponentų?
- ✓ Ar paliktas aktyvus root (administrator) vartotojas, kuris galėtų keisti VM paketo funkcionalumą?

Šiuo metu tarp populiariausių virtualizuotų operacinių sistemų atnaujinimo metodų galima paminėti tokį. Gamintojui išleidus VM paketo naują versiją atnaujinimas atliekamas tokiu būdų:

1) Išsaugoma (eksportuojama už VM ribų) statinė informacija
2) Sustabdoma senoji VM

3) Įkeliama ir paleidžiama naujos versijos VM
4) Įkeliama (importuojama) statinė informacija.

Tokiu būdu atnaujinama visa operacinė sistema ir jeigu senoji versija buvo pažeista arba joje buvo klastingos piktavališkos programinės įrangos pvz. „Rootkit" tipo, po tokio atnaujino ji tampa neaktyvi.

2.3. Hipervizorius ir piktavališka programinė įranga

Pirmiausia, kaip pats moderniausias yra vertas paminėjimo "Blue Pill" [23] atakos metodas tiesiogiai susijęs su virtualia infrastruktūra. Šiai atakai yra panaudojamos aparatūroje realizuotos virtualizacijos pagalbinės priemonės, kai įvykdžius atakos kodą pagrindinė operacinė sistema yra perkeliama į virtualizuotos operacinės sistemos lygmenį ir tarp aparatūros ir operacinės sistemos įterpiamas virtualizuojantis „rootkit" kodas. Tokios atakos atveju pagrindinė operacinė sistema nebetenka galimybės priešintis, nes visos instrukcijos ateinančios iš operacinės sistemos yra perimamos virtualizacijos lygmenyje ir gali būti modifikuojamos. Tokia įterpimo ataka gali būti realizuota ne tik įprastinei operacinei sistemai, bet ir I tipo hipervizoriui.

Tinklo paketai, keliaujantys tarp virtualių mašinų, keliauja uždarame virtualiame LAN tinkle ir yra neapsaugoti kriptografinėmis priemonėmis. Todėl tokia situacija turi kelias potencialias grėsmes:

- Sėkminga piktavalio prasiveržimo iki hipervizoriaus lygmens ataka leistų jam perimti ir modifikuoti visus virtualaus LAN tinklo paketus.
- Tradicinės tinklo saugos priemonės betarpiškai veikiančios tik su fizine aparatūra – IPS, IDS neturi galimybės tikrinti paketų keliaujančių virtualiais komutatoriais.

Dar viena grėsmė yra įprastinės DoS atakos realizacija virtualizuoje aplinkoje. Virtualios mašinos dažnai naudoja dinaminį resursų priskyrimą, todėl piktavalis, apkraudamas didžiausią prioritetą turinčią virtualią mašiną, automatiškai atima resursus ir iš likusių.

Ne taip seniai ataka, kai piktavalis išsiveržia iš virtualios mašinos į hipervizoriaus lygmenį ir iš jo atakuoja kaimyninę virtualią mašiną, buvo laikoma teorine. 2007 metų rudenį tai buvo pademonstruota praktiškai [17]. Atakos programinis kodas nebuvo paviešintas, tačiau galima spėti jog saugos analitikų ir piktavalių dirbančių šia linkme yra nemažai.

Ataka kai piktavalis perima hipervizoriaus kontrolę vadinama – hiperjacking, rezultate leidžia perimti visus duomenis einančius per hipervizoriaus lygmenį ir, priklausomai nuo piktavalio planų, juos keisti arba nukreipti piktavaliui reikalinga linkme.

Esant užgrobtam hipervizoriui ir neturint specialių saugiklių, virtualios mašinos neturi galimybės nustatyti jog hipervizoriaus apsauga buvo įveikta.

Kadangi hipervizorius yra operacinė sistema, jam nesvetimos “Rootkit“ tipo piktavališkos programinės įrangos įdiegimo grėsmės.

Tokios grėsmės tikimybė gana maža, nes hipervizoriaus kodas yra optimizuotas ir maksimaliai sumažintas, todėl pakeitus ar modifikavus kodo komponentus, pakeitimai būtų greitai pastebimi. Sėkmingas rootkit diegimas hipervizoriuje galimas dviem atvejais:

1) Prasiveržiant iš virtualizuotos OS į hipervizoriaus lygmenį ir įdiegiant rootkit elementus.
2) Perimant hipervizoriaus administravimo funkcijas ir įdiegiant rootkit elementus kaip įprastinę programinę įrangą pvz. hipervizoriaus įskiepius.

2.4. Virtualizuotose aplinkose egzistuojančios problemos

Pagal mokslininkų atliktą virtualių aplinkų analizę [9] labiausiai sutinkamos virtualizuotų informacinių sistemų problemos:

Nevaldomas augimas

Fizinių tarnybinių stočiu augimas organizacijose dažniausiai apribotas skiriamų lėšų ir įdiegimui reikalingo laiko. Visiška priešingybė yra virtualių mašinų kūrimas, nes ši operacija dažniausiai tėra rinkmenos nukopijavimas. Todėl sistemų vartotojai dažnai turi aibes virtualių mašinų, skirtų įvairiausiems tikslams – demonstracijoms, taikomųjų programų testavimui ar tiesiog taikomųjų programų kurių nebūna standartinėje kompiuterizuotoje darbo vietoje laikymui. Todėl bendras virtualių mašinų skaičius organizacijoje auga stulbinančiu greičiu, proporcingai erdvej turimai duomenų saugyklose.

Spartus virtualių mašinų kiekio augimas pradeda įtakoti organizacijų informacijos saugos sistemas. Labai retai būna taip, kad visos informacinių sistemų priežiūros užduotys pilnai automatizuotos – atnaujinimai, pataisymų valdymas, nustatymų valdymas, visa tai yra administratorių iniciatyvos ir atitinkamų įrankių kombinacija. Nevaldomai augantis virtualių mašinų kiekis gali ženkliai sumažinti administratorių pajėgumus ir atliekamų darbų kokybę, tai pat pasunkina saugos incidentų pasekmes. Pvz. Kirmino atakos metu operacinių sistemų kurias reikia išvalyti ir atnaujinti bus gerokai daugiau ir reakcijos laikas į incidentą bus gerokai ilgesnis.

Trumpalaikiškumas

Tradicinėse informacinėse sistemose vartotojas turi vieną arba dvi darbo stotis, kurios didžiąją dalį darbo laiko yra pasiekiamos. Išimtiniais atvejais vartotojai turi specialios paskirties, retai įjungiamas, darbo stotis arba į tinklą prijungia mobilią darbo stotį. Pradėjus naudoti virtualizacijos sprendimus, virtualių mašinų kiekis sukelia fenomenalią situaciją, kuri pasižymi tuo tinkle neprognozuojamai gali atsirasti arba dingti dideli kiekiai tinklo prisijungimų ir darbo stočių.

Jeigu įprastinius tinklus galima greitai ir paprastai atstatyti į „gerai žinomą“ konfigūraciją, tai dėl pastoviai besikeičiančio prisijungimų kiekio, to praktiškai neįmanoma padaryti tinkluose, kur naudojamos virtualios mašinos.

Pavyzdžiui, kirmino atakos atveju paprastai užkrečiami visi tinkle esantys kompiuteriai. Turėdamas tinkamus įrankius administratorius gana nesudėtingai gali nustatyti pažeistus kompiuterius, išvalyti kirminą ir užtaisyti spragas.

Virtualizuotose aplinkose tokia būseną pasiekti labai sunku, nes užkrėsti virtualūs kompiuteriai gali netikėtai ir trumpam atsirasti tinkle, užkrėsti sekančias aukas ir vėl būti išjungti prieš juos aptinkant. Arba kurį laiką nenaudotos ir neatnaujintos virtualios mašinos staiga įjungiamos ir darbo metu užsikrečia kirminu, vėliau jos išjungiamos ir sekančio įjungimo metu toliau platins kirminą. Rezultate mažo aktyvumo kirmino ataka gali tęstis ilgai kol bus nustatyti ir išvalyti visi užkrėsti tinklo mazgai.

Reikalavimus jog visos fizinės ir virtualios mašinos būtų įjungtos ir visos vienu metu atnaujintos sukuria priešpriešą tarp naudojamumo ir saugumo. Retai naudojamoms virtualioms mašinoms atnaujinti reiks ženkliai daugiau pastangų nei toms, kurios dirba kas dieną. Todėl dažniausiai tokios virtualios mašinos visai neatnaujinamos, tuo padidinant pažeidžiamų tinklo mazgų kiekį arba tiesiog atsakant jas naudoti, kas mažina virtualizacijos nešamą naudą.

Programinės įrangos gyvavimo ciklas

Tradicinėse sistemose programinės įrangos gyvavimo ciklą galima būtų pavaizduoti kaip tiesia liniją. Padedant įdiegimu ir sekant atnaujinimams, pataisymams ir konfigūracijos keitimams iki eksploatacijos pabaigos. Virtualizuotos sistemos gyvavimo ciklas labiausiai panašus į šakotą medį – bet kuris virtualios mašinos eksploatacijos momentas gali šakotis į keletą krypčių ir vienu metu gali egzistuoti kelios tos pačios sistemos kopijos.

Šį gyvavimo ciklo šakojimąsi įgalina virtualių mašinų grįžimo atgal į pradinę būseną funkcionalumai, kurie leidžia gražinti virtualią mašiną į pradinę būseną (pvz. klaidų paieška ir taisymas), arba kartoti programų vykdymą nuo išsaugoto laiko momento.

Tokia virtualizuotų sistemų eksploatacija kelią sunkumų tradicinėms atnaujinimų ir pataisymų valdymo sistemoms, kurios remiasi prielaida jog sistemos atnaujinimai ir pataisymai diegiami nuosekliai. Pavyzdžiui sugražinus virtualią mašiną į senesnę būseną, gali atsiverti saugos spragos, kurios buvo užtaisytos vėliau nei buvo išsaugota virtuali mašina, gali būti įjungiamos išjungtos vartotojų paskyros arba pakartotinai panaudoti kriptografiniai raktai. Taip pat gali būti pakarotinai paleidžiama piktavališka programinė įranga jei ji buvo aptikta ir išvalyti po būsenos išsaugojimo.

Įtaka saugos ir kriptografiniams protokolams

Būsenos išsaugojimo ir atstatymo funkcionalumas gali sujaukti daugelį šiandien naudojamų saugos protokolų. Pvz.:

- ✓ Vienkartinio slaptažodžio sistemos (pvz. S/KEY) kur informacija perduodama atviru tekstu ir sesijos saugumą garantuoja prielaida jog piktavalis nežino prieš tai buvusių apsikeitimo sesijų turinio. Jei virtuali mašina naudojanti S/KEY gražinama į ankstesnę būseną piktavalis gali peržiūrėti iš anksčiau sukauptus slaptažodžius.
- ✓ Protokolai kurie priklauso nuo atsitiktinių skaičių generavimo šaltinių pvz. įsivaizduokime virtualią mašiną kuri buvo gražinta į tokią būseną kai atsitiktinis skaičius jau sugeneruotas, bet dar nepanaudotas. Tokiu būdų „atsitiktinumas“, kuris turėtų būti nenuspėjamas ir neatkartojamas tampa nepatikimu.
- ✓ Srautiniuose šifravimo protokoluose, du skirtingi pranešimai gali būti užšifruoti tuo pačiu raktų srautu, taip išduodant piktavaliui abiejų pranešimų XOR duomenis, ko pasekoje gali būti atskleistas ir pranešimų turinys.

Pažeidžiami ir ne kriptografiniai protokolai pvz. TCP, gražinant virtualią mašiną į ankstesnę būseną galima nuspėti TCP sesijos pradinius paketų eiliškumo numerius ir taip realizuoti *TCP* perėmimo (hijacking) ataką.

Protokolai paremti paslapties žinojimu, jos neatskleidžiant, taip pat tampa pažeidžiami. Pvz.:

- ✓ Fiat-Shamirr arba Schorr autentifikavimo protokolai išduos vartotojo privatų raktą, jei ta pati vienkartinė reikšmė bus panaudota dar kartą
- ✓ Skaitmeninio parašo standartas (DSS) atskleis slaptą pasirašymo raktą jei du parašai bus sugeneruoti pasinaudojant ta pačia atsitiktine reikšme.

Todėl kriptografiniai mechanizmai paremti ir istoriniais vykdymo rezultatais, virtualiose aplinkose, tampa neefektyvūs ir tik didina sistemų darbo krūvį.

Įvairovė

Didžioji daugumą organizacijų stengiasi užtikrinti saugumą, naudojant identiškai įdiegtas ir atnaujinamas sistemų kopijas. Tačiau vienas iš virtualizacijos privalumų yra tas jog virtualizuojant galima naudoti senesnes, gal būt aparatūros nebepalaikomas, ir netaisytas operacines sistemas. Tokiu būdų sukuriama probleminė situacija kai bandoma išlaikyti tinkamą atnaujinimų lygį aibėje skirtingų operacinių sistemų, arba bandoma suvaldyti riziką kylančią iš nepataisytų operacinių sistemų.

Virtualios mašinos taip pat įtakojo ir programinės įrangos testavimo metodus. Ten kur anksčiau kiekvienam skirtingos operacinės sistemos testui reikėjo dedikuotos aparatūros, dabar programuotojas, turėdamas rinkinį virtualių mašinų su jam aktualiomis operacinių sistemų versijomis, gali atlikti testavimą savo darbo stotyje. Deja jei šios virtualios mašinos nebus prižiūrimos, jos gali tapti piktavališkos programinės įrangos židiniu.

Daugelyje kompanijų vartotojams pateikiant naują operacinę sistemą tiesiog duodama virtuali mašina į kurią palaipsniui perkeliamos taikomosios programos. Galima ir atvirkštinė situacija kai virtualioje mašinoje dirba senos ir reikalaujančios senesnės operacinės sistemos programos. Tokia praktika mažina perkėlimo ciklų sudėtingumą, tačiau veda į operacinių sistemų versijų daugėjimą. To pasekoje sudėtingėja atnaujinimų valdymo užduotys, ypač tais atvejais kai naudojamos senesnių versijų operacinės sistemos.

Padidėjęs mobilumas

Kol duomenų centro infrastruktūra buvo sudaryta iš grynai fizinių komponentų ir aparatūra nepajudinamai stovėjo spintose, tol serverio perkėlimo procesą iš spintos A į spintą B buvo galima laikyti valdomu procesu[17]. Tačiau atsiradus virtualizacijai ir tokioms technologijoms kaip vMotion ir Live migration, tai tampa nebeįmanoma. Virtualizacijos technologijos leidžia virtualią mašiną, jos nestabdant, perkelti į kitą fizinę mašiną, kuri tuo pat metu gali būti kitame tinklo segmente.

Todėl nebegalioja tradiciniai saugos perimetro įrankiai, kurie remiasi prielaidomis:

- ✓ Serveriai yra statinis elementas;
- ✓ Serverius lengva identifikuoti tinkle;
- ✓ Serveriai įprastai yra visą laiką įjungti.

Kadangi virtualių mašinų mobilumas tolygus paprastam failui[9], jas galima lengvai kopijuoti per tinklą arba nešiojamų laikmenų pagalba. Tai gali sukelti papildomas saugos problemas.

Įprastinėje aplinkoje patikimą skaičiavimų pagrindą (TCB - trusted computing base) sudaro aparatūra ir programinė įranga. Virtualizuotoje aplinkoje, patikimą skaičiavimų pagrindą sudaro visa įranga kurioje veikia virtuali mašina. Žinant, jog nėra kaupiama virtualios mašinos veiklos istorija, gali būti labai sudėtinga nustatyti saugos incidento pažeidimų mastą. Pvz. jei buvo pažeistas failų serveris, kuriame laikomos virtualios mašinos, yra tikimybė jog piktavalis jas modifikavo ir įdiegė slaptas duris (backdoor). Nustatyti ar VM failas buvo nukopijuotas ar perkeltas, gali būti sudėtinga.

Panašios problemos kyla virusų ir kirminų atakų atveju. Užkrėsti virtualią mašiną taip pat parasta kaip užkrėsti vykdomąjį failą. Tolimesnis, tiesioginis užkrėtimas suteikia galimybę skverbtis gilyn į operacinę sistemą, nepriklausomai nuo to kokią apsauginė programinė įranga įdiegta.

Virtualių mašinų naudojimas mobilioms darbo vietoms saugumo prasme nėra geriausias sprendimas, nes virtuali mašina be papildomos kontrolės gali būti perkeliama iš nekontroliuojamos pvz. namų aplinkos į kontroliuojamą organizacijos saugos perimetrą tuo padidinant atakos riziką.

Žiūrint iš fizinės saugos perspektyvos, virtualios mašinos būdamos failų pavidalu nėra apsaugotos nuo atviros vagystės. Įprastinių duomenų rinkmenų formatas ir galimybė, nešiotis virtualią mašiną nedidelėse laikmenose, leidžia vartotojui nešiotis didelius kiekius svarbios informacijos. Šio pavojaus rizika yra identiška nešiojamo kompiuterio vagystei.

Identifikavimas

Tradicinėse sistemose identifikacija dažnai siejama prie esamų operacinės sistemos arba aparatūros požymių – MAC adreso, vartotojų vardų, biuro adreso. Be identifikacijos būtų sudėtinga nustatyti atsakingus asmenis ir incidento atveju su jais susiekti.

Deja šie įprastiniai metodai visiškai netinka virtualioms mašinoms. Virtualių mašinų MAC adresai yra dinaminiai, tinklo prievadai gali keistis priklausomai nuo to kokioje fizinėje tarnybinėje stotyje virtuali mašina yra. Bandymas išjungti tinklo prievadą gali įtakoti kelias ar keliolika , nesusijusių su incidentu, virtualių mašinų dirbančių per tą patį prievadą.

Nustatyti atsakingus asmenis irgi gali būti sudėtinga, ypač tais atvejais kai virtuali mašina yra perduodama iš rankų rankas ir įvairiuose etapuose modifikuojama. Kadangi virtualia mašina nekaupia savininkų istorijos, nustatyti kokie pakeitimai ir kada buvo padaryti bus sudėtinga.

Duomenų gyvavimo ciklas

Vienas iš pagrindinių duomenų saugos principų – sumažinti jautrių duomenų sistemoje laikymo laiką. Virtualizacijos funkcionalumai sėkmingai įveikia ir šį principą. Virtualios mašinos būsenos išsaugojimas atima galimybę svečio operacinei sistemai sunaikinti jautrius trumpalaikius duomenis. Pvz. kriptografinius raktus. Duomenys, kurie turėtų būti ištrinti iškart po panaudojimo, kartais gali būti išlaikomi neribotą laiką, nes virtualią mašiną visada galima gražinti į išsaugotą būseną.

Duomenys išsaugomi už virtualios mašinos ribų - momentinių kopijų būsenos duomenys, darbinės atminties išsaugojimo rinkmenos, taip pat kelia papildomą riziką ir apeina operacinės sistemos mechanizmus, kurie turėtų užtikrinti laikinų duomenų sunaikinimą.

Viso to pasekoje, laikini tačiau svarbūs duomenys, gali likti pagrindinėje virtualios mašinos operacinėje sistemoje neribotą laiką.

Virtualizuotos operacinės sistemos taip pat kaip ir įprastu būdu dirbančios operacinės sistemos dažnai yra atakuojamos virusų, kirminų arba piktavalių asmenų. Sėkmingų atakų atveju sistemoje paleidžiami parazitiniai servisai (Spam SMTP serveriai, DDoS agentai) arba įdiegiamos paslėptos durys (angl. *backdoor, rootkit*). Šias piktavalių programas įvairiais būdais stengiamasi paslėpti nuo administratorių ir OS saugos priemonių – antivirusų, OS vientisumo analizatorių. Kaip slėpimo priemones reiktų paminėti:

- ✓ procesų stebėjimo įrankių modifikavimas (ps, top, taskmgr.exe)
- ✓ failų valdymo įrankių modifikavimas, kad nerodytų tam tikrų failų (ls, dir)
- ✓ antiviruso procesų stabdymas ir savęs užšifravimas aptikus aktyvias saugos priemones pvz. kai paleidžiamas antiviruso failų skanavimo procesas.

Skirtingai nuo įprastinės aparatinės aplinkos virtuali aplinka gali palengvinti tokio tipo kenkėjiškų programų aptikimą. Pagrindiniai virtualios aplinkos privalumai aptinkant kenkėjišką programinę įrangą:

- ✓ nėra būtina leisti derinimo ar analizavimo procesus virtualizuotoje operacinėje sistemoje, nes galime peržiūrėti virtualios mašinos rinkmenas ir atmintį pasinaudodami hipervizoriumi ar pagrindine operacine sistema.
- ✓ Kadangi visa aptikimo ir diagnostikos programinė įranga leidžiama už tiriamos operacinės sistemos ribų, mažesnė tikimybė jog piktavališka programinė įranga persijungs į maskavimosi režimą.
- ✓ Galima, momentinių kopijų pagalba, išsaugoti skirtingus virtualizuotos operacinės sistemos darbo etapus ar būsenas ir vėliau juos individualiai analizuoti.
- ✓ Galima išsaugoti virtualios mašinos būklę prieš imantis aktyvių veiksmų, kurie gali iššaukti duomenų ar piktavališkos programinės įrangos pėdsakų ar kodo sunaikinimą.
- ✓ Galima paprastai ir patogiai daryti virtualių mašinų kopijas ir jas padalinti kiekvienam incidento tyrėjui individualiai

Nevaldomas virtualizuotų serverių kiekio augimas

Atsiradus virtualizacijai sukurti naują virtualų serverį tapo labai lengva, daugeliu atvejų; tiesiog reikia nukopijuoti rinkmeną – tai leidžia verslui efektyviai ir greitai pateikti paslaugas į rinką ar bandyti naujus produktus. Tačiau eksploatuojančiam ir informacijos saugos personalui tokia galimybė kelia daugybę rūpesčių, nes kiekviena naujai įdiegta operacinė sistema tai papildomas saugos įvertinimas, OS atnaujinimo vienetas, OS stebėjimo vienetas.

Susidarius tokiai situacijai atsiranda būtinybė valdyti virtualių mašinų gyvavimo ciklą, nuo sukūrimo iki sunaikinimo.

Apsaugos priemonės ir virtualizuotos informacinės sistemos

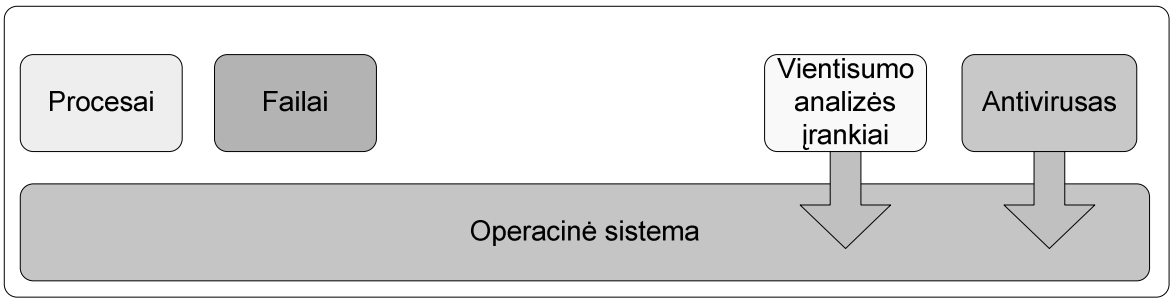
Daugelis tradicinių saugos priemonių gamintojų perkėlė savo informacijos saugos produktus į virtualias mašinas, tačiau tai dar nereiškia, kad tos priemonės tapo tinkamos virtualiai infrastruktūrai. Kaip pavyzdį paimkime Antivirusinę programinę įrangą. Dėl neprognozuojama virtualių mašinų gyvavimo ciklo – negalėsime tiksliai suskaičiuoti reikalingų licencijų kiekio. Jei ataka bus realizuota iš hipervizoriaus lygmens, antivirusinė

programinė įranga įdiegta virtualioje mašinoje to nepastebės. Palyginkime tradicinės ir virtualizuotos operacinės sistemos saugos analizės modelius.

2.5. OS saugos analizės modeliai

OS saugos analizės modelius galime klasifikuoti pagal tai kur dirba saugos analizės įrankiai.

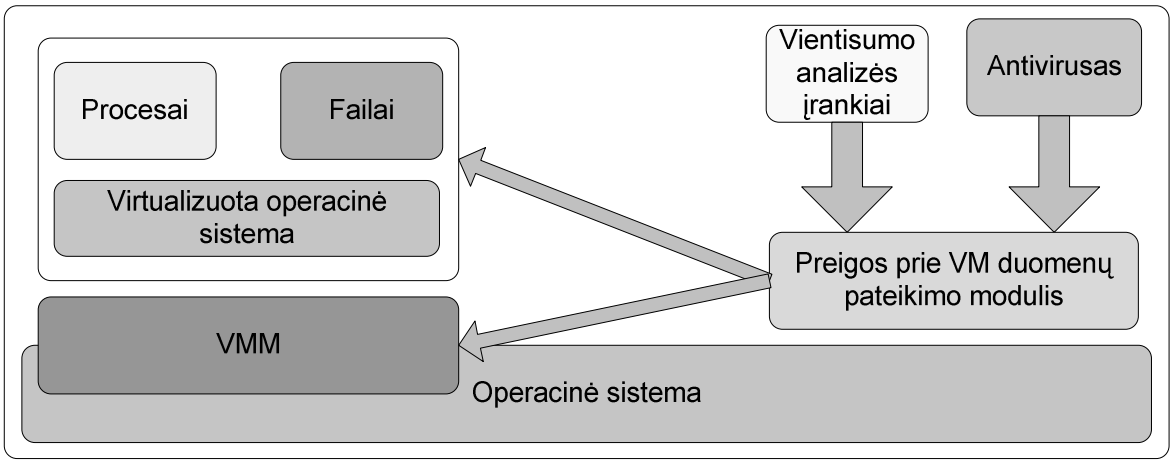
- ✓ Įprastiniame modelyje saugos priemonės dirba toje pačioje operacinėje sistemoje, kuri tuo metu yra analizuojama.



3 pav. – Įprastinis OS saugos analizės modelis

Šio modelio trūkumas yra tas, kad kenkėjiškos programos gali tiesiogiai įtakoti saugos įrankių darbą (pvz. sustabdyti antivirusinę programinę įrangą).

- ✓ Virtualizuotame modelyje saugos priemonės dirba virtualizuotos operacinės sistemos išorėje todėl negali būti įtakojamos, kenkėjiškų programų.
  Taip pat kenkėjiškos programos negali aptikti veikiančių saugos priemonių.



4 pav. – Virtualizuotas OS saugos analizės modelis

Vienintelis šio modelio trūkumas, kad kenkėjiškos programos gali nustatyti ar jos dirba virtualizuotoje operacinėje sistemoje ir imtis papildomų maskavimosi priemonių.

2.6. Piktavališkos programinės įrangos aptikimo įrankiai

Šio tyrimo tikslas išanalizuoti esamus piktavališkos programinės įrangos virtualiose aplinkose aptikimo kryptis ir modelius ir pasiūlyti savo variantą.

Šiuo metu informacijos šaltiniuose aptinkamos šios pagrindinės piktavališkos programinės įrangos virtualiose aplinkose aptikimo kryptys:

- ✓ Hipervizoriaus ir jo konfigūracijos auditas
- ✓ Hipervizoriaus prasiskverbimo testai
- ✓ Virtualios mašinos kaip pirminio atakos šaltinio prasiskverbimo testai
- ✓ Virtualių mašinų gyvavimo ciklo ir automatizacijos sistemos
- ✓ Į hipervizorių integruota apsauga
- ✓ Virtualios mašinos su saugos funkcionalumu
- ✓ Atviras virtualizacijos formatas
- ✓ Hipervizoriaus saugos įskiepiai (plug-ins)
- ✓ Saugos agentai
- ✓ IDS sistemos grįstos konfigūracijos failų analize

Hipervizoriaus prasiskverbimo testai

Šių testų atveju atakuojamas pats hipervizorius arba jo valdymo terminalas tikintis sutrikdyti darbą arba perimti valdymą. Šios atakos yra sudėtingos, nes hipervizoriaus kodas yra labai mažos apimties ir uždaras.

Virtualios mašinos kaip pirminio atakos šaltinio prasiskverbimo testai

Tai toks saugos audito metodas, kai įvairiu kritines situacijas sukeliančiu kodu atakuojamas hipervizorius arba jo valdymo priemonės, tikintis iš virtualios mašinos paveikti visą fizinę tarnybinę stotį ar net visą virtualią infrastruktūrą[20].
Dažniausiai yra intensyviai siunčiamos klaidingos procesoriaus ir atminties valdymo komandos, tikintis, jog pavyks sutrikdyti CPU arba įvedimo/išvedimo valdiklių darbą arba

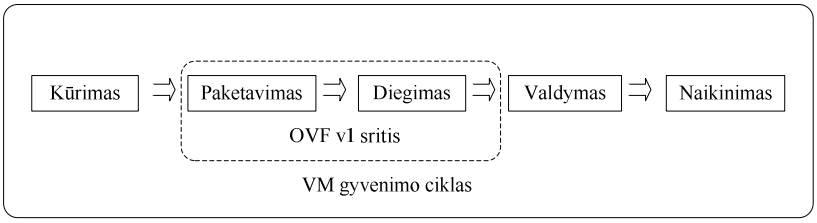
priversti sistemos klaidų tikrinimo ir šalinimo funkcijas dirbti taip aktyviai, jog normalus darbas tampa nebeįmanomas.
Populiarūs tokių testų įrankiai – CRASHME, IOFUZZ

Virtualių mašinų gyvavimo ciklo ir automatizacijos sistemos

Šis metodas orientuotas į Virtualių Mašinų gyvavimo ciklo valdymą, nuo sukūrimo iki sunaikinimo. Tokiu būdu yra užtikrinama jog virtualizuotoje infrastruktūroje nebus virtualių mašinų dublikatų, kurių lygiagretus veikimas gali sukelti kritines pasekmes. Taip pat užtikrina jog nebus apleistų ir neatnaujintų virtualių mašinų, kurias savo veiksmams galėtų panaudoti piktavaliai.

Atviras virtualizacijos formatas

Nuo 2007 metų rudens rinkoje naudojamas Atviras Virtualizacijos Formatas (Open Virtualisation Format) – 5 pav. Jo kūrimo darbo grupę sudarė Dell, HP, IBM, Microsoft, VMware ir XenSource atstovai. Formatas XML pagalba aprašo modelį, kurio pagalba aprašoma virtualioji mašina.
Formatas buvo sukurtas su tikslu aprašyti programinės įrangos platinimo virtualių mašinų paketais modelį.



5 pav. Programinio produkto platinamo virtualios mašinos paketo pavidalu gyvavimo ciklas ir OVF nagrinėjama sritis

Į hipervizorių integruota apsauga

Visiškai įmanoma integruoti saugos mechanizmus į hipervizoriaus kodą, tačiau tai nėra populiarus sprendimas dėl dviejų priežasčių:

- ✓ Hipervizoriaus gamintojas turi perkelti gamybos dėmesį nuo hipervizoriaus technologinio vystymo į jo saugos vystymą, kas sudarytų sąlygas konkurentams aplenkti gamintoją našumo ir funkcionalumo srityje.
- ✓ Įskiepių naudojimas padidintų hipervizoriaus kodo apimtis ir taip padidėtų galimų atakų tikimybė

Hipervizoriaus saugos įskiepiai

Kaip jau buvo minėta, hipervizorių gamintojai nenoriai perkelia produkto vystymo dėmesį į saugumo sritis, tačiau įskiepių API pagalba leidžia tą daryti kitiems gamintojams. Kaip pavyzdį galima pateikti VMware VMSafe API, kuris leidžia:

- ✓ vykdyti tinklo stebėjimą ir tinklo paketų modifikavimą,
- ✓ vykdyti komandas virtualių mašinų viduje, pasinaudojant sisteminėmis komandomis
- ✓ valdyti virtualių mašinų atmintį ir joje veikiančius procesus.

Tačiau visos minėtos savybės gali būti panaudotos tiek geriems tiek piktavališkiems tikslams.

Saugos agentai

Vienas iš galimų saugos metodų – saugos agentų diegimas į virtualias mašinas, o centrinio valdymo terminalas diegiamas atskiroje tarnybinėje stotyje. Deja šiai dienai virtualizacijos sprendimų gamintojai neturi metodikos, kaip tokio tipo saugos produktai turi būti sertifikuojami virtualizuotoms informacinėms sistemoms.

Hipervizoriaus ir jo konfigūracijos auditas

Šio sprendimo pagrindas hipervizoriaus ir jame veikiančių virtualių mašinų nustatymų analizė.

Dauguma auditavimo priemonių pradinių analizės duomenų surinkimui naudojasi gamintojo pateiktais programiniais komponentais.

Surinkti duomenys vertinami lyginant situaciją su gamintojo ir saugos standartų rekomenduojamais nustatymais.

IDS sistemos grįstos hipervizoriaus konfigūracijos failų analize

Šiuo metu rinkoje sutinkamos šios automatizuotos virtualios infrastruktūros auditavimo sistemos:

- ✓ Tripwire – ConfigCheck
- ✓ Configuresoft Inc. - Compliance Checker for VMware ESX

Tripwire – ConfigCheck

Tripwire ConfigCheck yra nemokamas įrankis sukurtas bendradarbiaujant su kompanija VMware, skirtas automatizuotam virtualios infrastruktūros auditavimui, pagal VMware

saugos rekomendacijų dokumentą - VMware Infrastructure 3 Security Hardening guidelines. ConfigCheck įrankis padeda:

- ✓ surasti neteisingai sukonfigūruotus komponentus
- ✓ surasti potencialias saugos spragas
- ✓ palengvina saugos priemonių diegimą ir suderinamumą su saugos standartais

Configuresoft Inc. - Compliance Checker for VMware ESX

Compliance Checker for VMware ESX taip pat yra nemokamas įrankis skirtas audituoti virtualias infrastruktūras, veikiančias VMware ESX pagrindu. Leidžia atlikti virtualios infrastruktūros saugos įvertinimą pagal du saugos modelius:

- ✓ VMware Infrastructure 3 Security Hardening guidelines
- ✓ CIS benchmarks for ESX.

Atlikus auditą rezultatus galima išsaugoti kaip ataskaitą. Unikalus Compliance Checker for VMware ESX funkcionalumas yra tas jog galima išsaugoti auditų istorija ir ją naudoti kaip pagrindą palyginimui ateityje.

Pastarųjų sistemų pagrindinis trūkumas tas jog analizuojami tik hipervizoriaus ir jo valdymo programinės įrangos konfigūraciniai failai.

2.7. Klastingos piktavališkos programinės įrangos klasifikacija

Šiame skyriuje detaliau apžvelgsiu klastingos piktavališkos programinės įrangos klasifikaciją pagal tai kokius maskavimosi metodus ji naudoja, kokiame operacinės sistemos lygmenyje veikia ir pagal tai kokį pažangumo lygį yra pasiekę šios programinės įrangos kūrėjai.

2.7.1. Pagal maskavimosi metodus

Yra išskiriami keturi pagrindiniai klastingos, piktavališkos programinės įrangos tipai[24], juos plačiau ir apžvelgsiu:

- ✓ Tiesioginis maskavimas – piktavališka programinė įranga, kuri apsimeta esant įprastine (normalia) programine įranga, kaip pavyzdžiui komanda *dir*, kuri iš tiesų NErodo katalogo turinio.
- ✓ Paprastasis maskavimas – programinė įranga pateikiama taip, kad vartotojas ja pasinaudoja negalvodamas apie saugumą (pvz. programa - sex)

- ✓ „Slidus“ maskavimas – piktavališkos programinės įrangos komponentui (komandai) suteikiamas vardas panašus į įprastinės sistemos komandos. Pvz. dr – paleidžia rootkit, o dir - parodo katalogo turinį.
- ✓ Maskavimasis aplinkoje – piktavališka programinė įranga sunkiai atskiriama nuo įprastinės programinės įrangos. Pvz. atlieka reikalingą funkciją, tačiau lygiagrečiai paleidžia procesus su „šalutiniu poveikiu“.

2.7.2. Pagal operacinės sistemos lygius

Pagal tai kokiame operacinės sistemos lygyje ar režime dirba piktavališka programinė įranga, ją galima suskirstyti į dvi kategorijas [26] – vartotojo režimo ir branduolio režimo.

Vartotojo režimo piktavališka programinė įranga

Vartotojo režimo arba dar vadinama programinio lygio „rootkit“ programinė įranga.

Pakeičia kritines sistemos komandas (tokias kaip ls, ps, netstat) jų modifikuotomis versijomis, kurių pagalba paslepiami piktavališki OS procesai arba tinklo prisijungimai. Kadangi vartotojo lygio „rootkit“ nemodifikuoja OS branduolio, mes galime sulyginti informaciją gaunamą tiesiai iš branduolio su informacija gaunama iš modifikuotų komandų. Tokiu būdų pastebėjus skirtumą tarp pateikiamos informacijos galima aptikti vartotojo lygio „rootkit“ ataką. Šio tipo atakos yra lengviau aptinkamos minėtu būdu arba reguliariai tikrinant sistemos komponentų kontrolines sumas (MD5 ar pan.)

Pagal saugos žiedų koncepciją vartotojo režimo rootkit veikia trečiajame žiede 6 pav. - Ring3 [t.v. userland]. Trečiasis žiedas tai ta aplinka kurioje veikia vartotojo programos. Kadangi šiame žiede visos programos laikomos nepatikimomis ir veikia žemiausiomis privilegijomis, piktavališkos įrangos aptikimas palyginus su branduolio lygio piktavališka programine įranga yra gerokai paprastesnis.. Vartotojo režimo rootkit modifikuoja procesus, tinklo prisijungimus, duomenų rinkmenas ir įvykius. Tam tikros aptikimo metodikos gali turėti sunkumų aptinkant vartotojo režimo rootkit, vien todėl kad ne visada aišku kas yra piktavalio procesas, o kas ne. Dažniausiai taip yra dėl to kad falsifikuojama visa informacija galinti padėti nustatyti piktavališkus procesus sistemoje.

Vartotojo režimo rootkit aptikimo metodikas galima suskirstyti į šias kategorijas:

- ✓ Euristinę analizę
- ✓ Anomalijų analizę
- ✓ Parašų analizę
- ✓ Sulyginamąją analizę (cross-view)

Tačiau tai dar nereiškia jog vartotojo režimo „rootkit“ aptikimas yra paprastas. Slaptumas yra pagrindinė ir vienintelė „rootkit“ savybė leidžianti piktavališkam procesui išlikti sistemoje išlaikant administratoriaus privilegijas. „Rootkit“ naudojasi administratoriaus privilegijomis tam, kad veikti sistemoje tačiau tai nėra įrankis šioms privilegijoms pasiekti. Tai reiškia jog prieš įdiegiant „rootkit“ komponentus piktavalis turi įveikti operacinės sistemos apsaugas ir sistema buvo priversta piktavalio privilegijas paaukštinti iki administratoriaus, tik po to galimas rootkit komponentų diegimas.

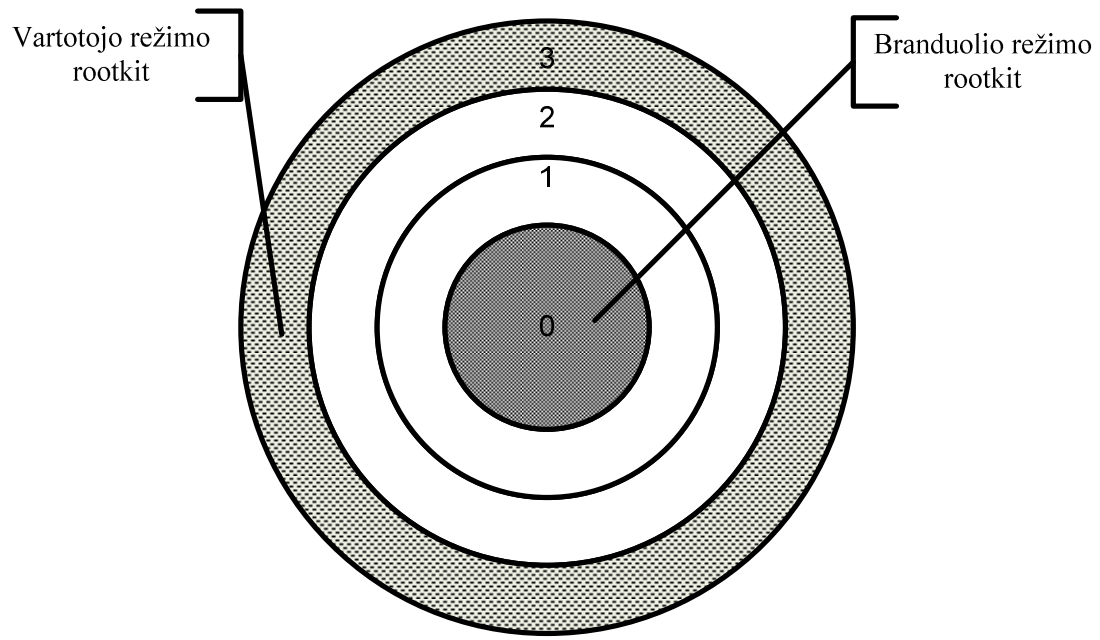
Vartotojo režimo piktavališkos programinės įrangos naudojami maskavimosi būdai

Vartotojo režimo „rootkit“ naudoja aibę metodų tinkamų pasislėpti nuo aptikimo. Dažniausiai naudojami:

- ✓ Procesų slėpimas
- ✓ Branduolio modulių arba bibliotekų injekcijos
- ✓ Registry šakų modifikavimas
- ✓ Rinkmenų modifikavimas
- ✓ Sisteminių lentelių įrašų modifikavimas

Dažniausiai naudojami visi paminėti metodai, tam, kad kuo ilgiau išlikti nepastebėtiems. Su kiekviena piktavališkos programinės įrangos karta aptikimas darosi vis sudėtingesnis.

Šiame kontekste toliau ir panagrinėsiu įvairias metodus tinkamus vartotojo režimo rootkit procesams paslėpti ir aptikimo mechanizmus naudojamus jiems aptikti.



6 pav. Saugos žiedai ir skirtingų režimų piktavališka programinė įranga

Branduolio režimo piktavališka programinė įranga

Šio tipo atakos aptinkamos gerokai sunkiau ir įkūnija nesibaigiančia kovą tarp piktavalių ir informacijos saugos ekspertų. Tam, kad pateikti suklastotą informaciją apie sistemą, branduolio lygio „rootkit“ modifikuoja OS branduolį. Tokios atakos atveju jeigu sisteminės komandos ir yra nemodifikuotos, jos vis tiek nepateikia teisingos informacijos. Rezultate visi bandymai nustatyti įsibrovimą lieka bevaisiai.

Branduolio rootkit procesai veikia operacinės sistemos branduolyje – 6 pav., kas apsunkina jų aptikimą. Dažniausiai, procesų slėpimo eigoje, modifikuojama visa operacinė sistema tam kad paslėpti bet kokius „rootkit“ egzistavimo ir sistemos sukompromitavimo pėdsakus.

Branduolio tipo „rootkit“ piktavališka programinė įranga yra laikoma pačia pavojingiausia, nes stipriai pakenkia sistemos vientisumui ir savidiagnozės galimybėms.

Žiūrint iš esmės, bet kuris įrankis skirtas tokiai programinei įrangai aptikti yra imlus sistemos signalų duomenų falsifikavimui. Kai kurie įrankiai, taip pat ir įrankis naudojamas

šiame tyrime, gali dirbti už pažeistos sistemos ribų, kas padidina atsparumą sistemos signalų klastojimui ir gerokai padidina aptikimo tikimybę.

Sistemos saugai apgauti dažniausiai naudojami šie metodai:

- ✓ Tiesioginis sistemos signalų lentelės modifikavimas
- ✓ Sistemos signalų šuoliai
- ✓ Pertraukimų aprašymo lentelės (IDT) modifikavimas
- ✓ IA32 derinimo režimas (IA32 Debug)

Branduolio „rootkit“ pagal žiedų terminologiją:

Pagal žiedų terminologiją branduolio rootkit veikia nuliniame žiede - ring0 (6 pav.). Kadangi nulinis žiedas yra po visais kitais žiedų sluoksniais, branduolio rootkit stebėjimas iš tos pačios sistemos tampa neįmanomu. Žinoma yra išimčių, pavyzdžiui metodikos grįstos būsenų sulyginimais arba įvairių sistemos signalų užklausų siuntimas ir pastovus atsakymų į jas lyginimas.

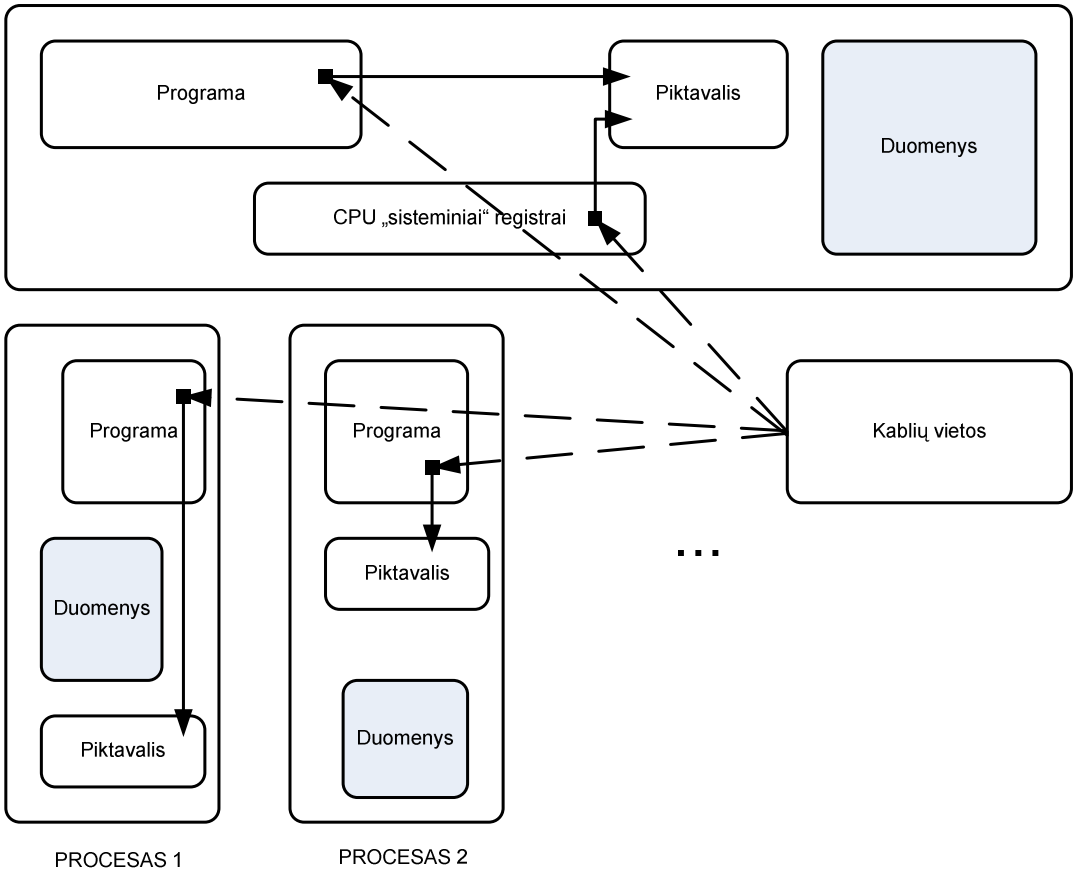
Tačiau prisimenant, jog branduolio „rootkit“ modifikuoja didžiąją dalį operacinės sistemos savybių ir funkcijų, kurias galima būtų naudoti aptikimui, operacinės sistemos analizė tampa beveik maksimaliai sudėtinga.

Kaip jau minėta pagal žiedų koncepciją nulinis žiedas turi aukščiausias privilegijas, o trečiasis mažiausias. Dauguma rootkit aptikimo sprendimų paleidžiami iš vartotojo aplinkos todėl patenka į trečiojo žiedo sritį ir jeigu piktavališką programinė įranga dirba taip kaip numatė piktavalis, jos aptikimas tampa sudėtingas.

2.7.3. Klastingos piktavališkos programinės įrangos klasifikacija pagal pažangumo lygį

Pagal pažangumo lygį piktavališką programinę įrangą galima suskirstyti į keturis tipus – nulinio, pirmo, antro ir trečio.

Nulinio tipo piktavališka programinė įranga nenaudoja nedokumentuotų operacinės sistemos funkcijų ar komponentų, saugai pažeisti ar atakoms vykdyti, todėl žiūrint iš sistemos saugos pažeidimo pozicijų yra mažiausiai dominanti.

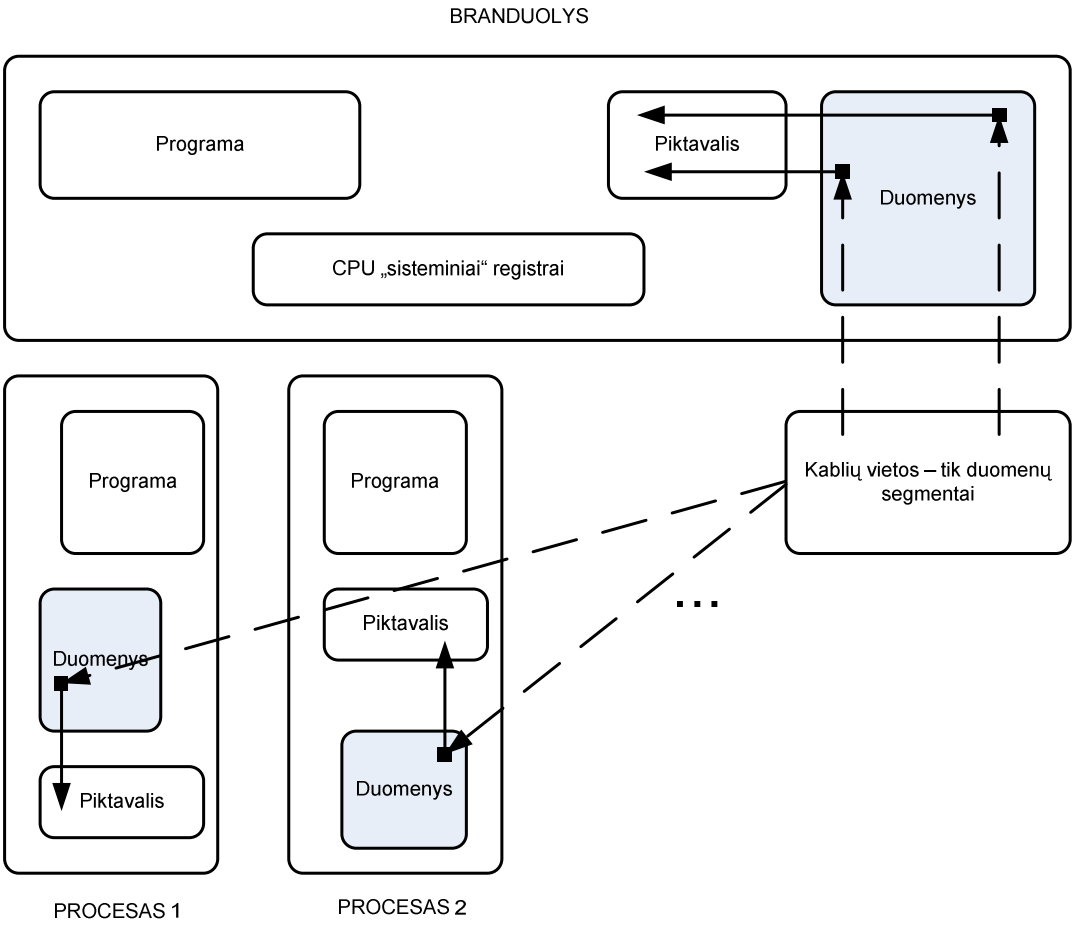


7 pav. Pirmo tipo piktavališka programinė įranga

Pirmo tipo (7 pav.) piktavališka programinė įranga pažeidžia ir maskuojasi veikiančių programų kode, toks metodas labai savo elgsena ir požymiais panašus į įprastinį viruso užkratą todėl yra lengvai aptinkamas. Jį dažniausiai išduoda modifikuoti paleidžiamieji failai. Veiksmingiausias metodas aptikimui – reguliarus paleidžiamųjų failų vientisumo tikrinimas. Vientisumo tikrinimą galima atlikti reguliariai tikrinant visų kritinių rinkmenų kontrolines sumas arba tikrinant paleidžiamųjų rinkmenų kriptografinius parašus. Pastarąjį metodą jau kuris laikas naudoja Microsoft Windows aplinkoje veikiančių taikomųjų programų kūrėjai. Taip pat tokio tipo piktavališką programinę įrangą nesunkiai aptinka antivirusinė programinė įranga.

Antro tipo piktavališka programinė įranga maskuojasi dirbančių programų duomenų segmentuose. Visa šio metodo esmė tokia, kad duomenys, kurie yra modifikuojami piktavališkos programinės įrangos, yra modifikuojami ir normaliomis sąlygomis. t.y. duomenis modifikuoja pati operacinė sistema arba taikomoji programa priklausomai nuo to, kuris iš šių komponentų yra pažeistas. Tai reiškia jog duomenims pastoviai keičiantis,

vientisumo patikrinimas tampa neįmanomas arba netikslus, nes skaičiuoti kontrolines sumas duomenų segmentams būtų beprasmiška

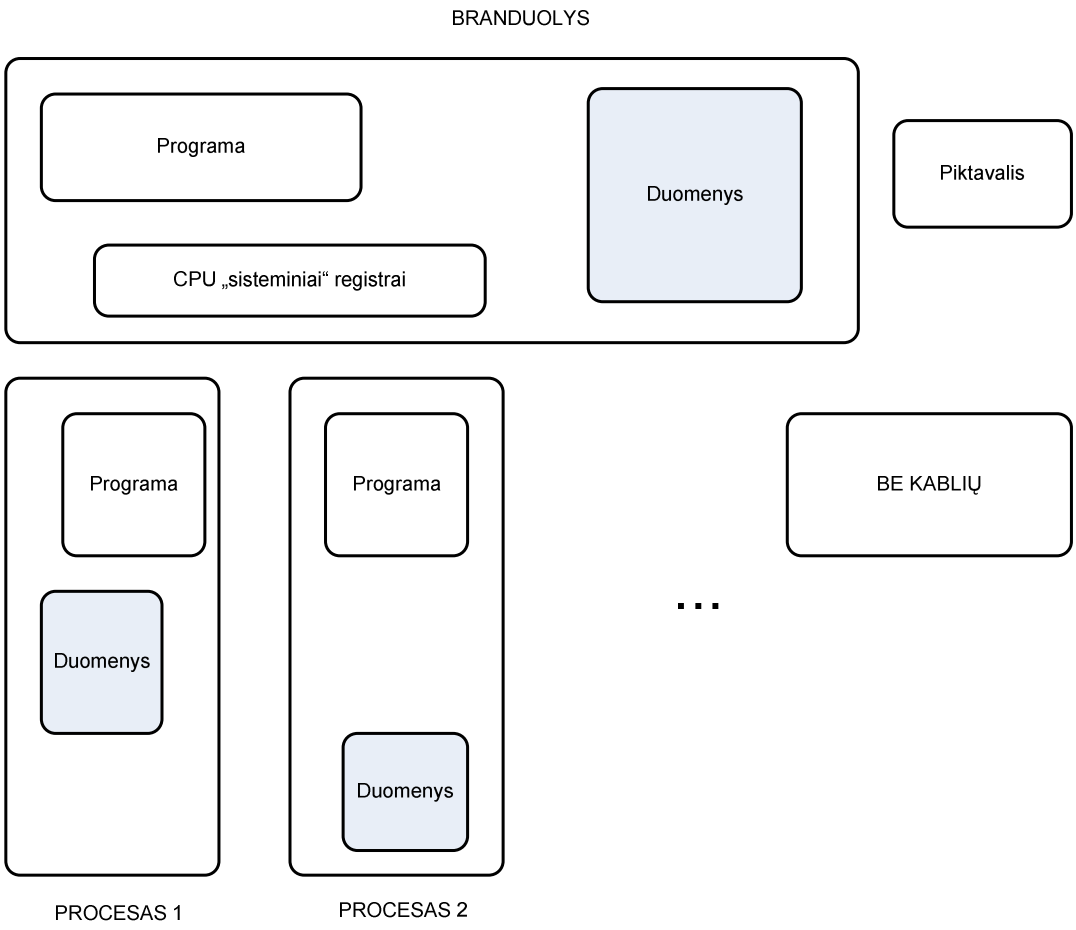


8 pav. Antro tipo piktavališka programinė įranga

.

Tam, kad aptikti II tipo piktavališką programinę įrangą turėtume išanalizuoti visus operacinės sistemos ir taikomųjų programų duomenų segmentus, taip pat ir tvarkykles ir operacinės sistemos saugos posistemės komponentus. Kiekviename iš paminėtų komponentų turėtume surasti operacinės sistemos saugai kritinius elementus ir juos tikrinti kiekvienos analizės metu.

Šiuo metu, veikiančioje sistemoje, aptikti II tipo piktavališką programinę yra sudėtinga, nes šiuolaikinės operacinės sistemos pagal savo realizaciją yra netinkamos veikiančių komponentų palyginamajam tikrinimui.



9 pav. Trečio tipo piktavališka programinė įranga

Trečias piktavališkos programinės įrangos tipas (9 pav.), nenaudoja sisteminių lentelių įrašų funkcijų ir nėra susietas su operacinės sistemos ar taikomųjų programų segmentais, todėl jo aptikimas yra dar sudėtingesnis nei antrojo tipo. Jeigu ir atrastume metodą, kaip palyginamuoju būdu aptikti antro tipo piktavališką programinę įrangą jos negalėtume pritaikyti trečiajam tipui, nes trečio tipo piktavališka programinė įranga veikia už operacinės sistemos ribų. Šis metodas yra realizuotas vadinamoje „BluePill“ [23] atakoje, kai pasinaudojant aparatūroje esančiomis virtualizacijos pagreitinimo instrukcijomis, dirbanti informacinė sistema yra paverčiama I tipo virtualizacijos realizacija, o piktavališkas kodas dirba hipervizoriaus lygmenyje.

2.8. „Rootkit“ tipo piktavališkos programinės įrangos įdiegimas ir veikimas

Kad sėkmingai įdiegti „rootkit” į operacinę sistemą, piktavalis sėkmingai turi atlikti šiuos veiksmus:

1) Patalpinti „rootkit” kodą į branduolį
2) Užmaskuoti „rootkit” pėdsakus
    a. Modifikuoti OS sistemines užklausas
    b. Modifikuoti OS branduolio sistemines lenteles
    c. Modifikuoti OS branduolio kodą
    d. Modifikuoti kritinius OS branduolio komponentus.

10 pav. Piktavališkos programinės įrangos įsiskverbimo ir maskavimosi būdai

2.9. Virtualizacija ir klastinga piktavališka programinė įranga

Pagrindinė Virtualizuotose aplinkose piktavališka programinė įranga išlaiko savo funkcijas ir kartais netgi pasinaudoja virtualizavimo mechanizmais tam, kad dar efektyviau paslėpti savo egzistavimą[23]. Analizės metu nebuvo pastebėta jokių technologinių sprendimų, kurie sustabdytų piktavališką programinę įrangą vien perkėlus operacinę sistemą iš aparatinės aplinkos į virtualizuotą aplinką.

2.10. Antivirusinės programinės įrangos naudojimo efektyvumas virtualizuotoje aplinkoje

Antivirusinę programinę įrangą sėkmingai galima naudoti virtualizuotose aplinkose. Tačiau priklausomai nuo antivirusinės programinės įrangos įdiegimo skirsis jos efektyvumas. Apžvelgsiu pagrindinius antivirusinės programinės įrangos virtualizuotose aplinkose naudojimo ypatumus.

Virtualios mašinos viduje veikianti antivirusinė programinė įranga veiks lygiai taip pat kaip ir būdama aparatinėje aplinkoje veikiančioje operacinėje sistemoje. Tai reiškia jog bus pažeidžiama „protingos“ piktavališkos programinės įrangos poveikiui [14]. Kaip pavyzdį galima paminėti Agobot, piktavališką programinę įrangą, kuri sugeba aptikti ir išjungti 105 antivirusinės ir kitokias apsaugos funkcijas vykdančios programinės įrangos procesus.

Antivirusinei programinei įrangai veikiančiai virtualios mašinos viduje dažniausiai galioja standartinės licencijavimo taisyklės – viena licencija vienam kompiuteriui. Žinant specifinį virtualių mašinų dinamiškumą ir trumpaamžiškumą, tokia licencijavimo tvarką gali tapti kliūtimi siekiant efektyviai veikiančios informacinės sistemos.

Virtualios mašinos išorėje veikianti antivirusinė programinė įranga bus gerokai efektyvesnė nes:

- ✓ gali lengviau išvengti piktavališkos programinės įrangos maskavimosi ir gynybinių veiksmų t.y. bus sunkiau apgauti sensorius arba juos išjungti.
- ✓ Paprasčiau licencijuojama, nes licencijos reiks tik pagrindinei sistemai kurioje veikia hipervizorius, o tai dažniausiai būna ilgalaikė sistema.

2.11. Analizės išvados

- ✓ Virtualizacija ne tik palengvina sistemų valdymą ir leidžia efektyviau išnaudoti aparatūros resursus, bet ir sukuria daug įvairių informacinių sistemų saugos iššūkių.
- ✓ Piktavališkos programinės įrangos atakos išlieka aktualios virtualiose aplinkose
- ✓ Virtualizuotose aplinkose piktavališka programinė įranga išlaiko savo funkcijas ir kartais netgi pasinaudoja virtualizavimo mechanizmais tam, kad dar efektyviau paslėpti savo egzistavimą.
- ✓ Darbo analizės metu buvo nustatyti pagrindiniai informacijos saugos iššūkiai, kuriuos sukuria virtualizacijos sprendimai.

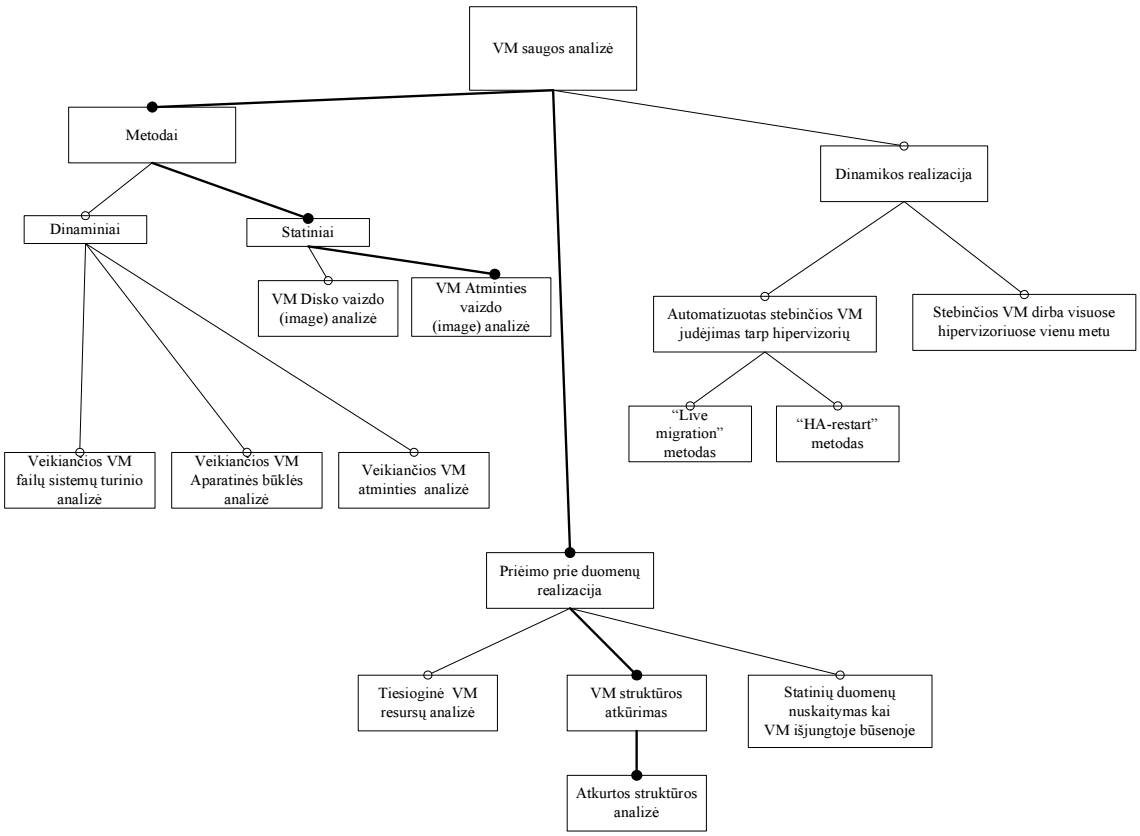
- ✓ Piktavališką programinę įrangą nagrinėjantys straipsniai dažniausiai, analizuoja tradicines, o ne virtualizuotas informacines sistemas
- ✓ Buvo nustatyta jog virtualizuotose aplinkose tradicinių saugos analizės priemonių veiksmingumas ribotas arba reikalauja papildomų pastangų.
- ✓ Tam, kad efektyviai kovoti su piktavalių kuriama klastinga programine įranga reikia ištirti kokios priemonės tam tinkamiausios.

3. Klastingos piktavališkos programinės įrangos aptikimo metodika.

Kadangi kenkėjiška programinė įranga sugeba pasislėpti nuo tradicinių priemonių, kuriami nauji apsaugos metodai. Čia mums gali padėti virtualizacija, kurios pagalba išoriškai (slapta) galime stebėti operacinės sistemos darbą, neaktyvuojant kenkėjiškos programinės įrangos savisaugos funkcijų.

Saugos analizatorius naudodamas hipervizoriaus atminties ir procesoriaus valdymo priemones gali nepastebimai analizuoti operacinės sistemos žemo lygio funkcijas, tikrinti atmintį ir procesoriaus vykdomas užduotis.

Kaip vienas iš veiksmingiausių virtualių mašinų metodų buvo pasirinktas virtualios mašinos darbinės atminties tyrimas „iš išorės“. Jo pagrindu buvo sudaroma metodika ir atliekamas tyrimas.



11 pav. - Saugos analizatoriaus realizacijos savybių diagrama

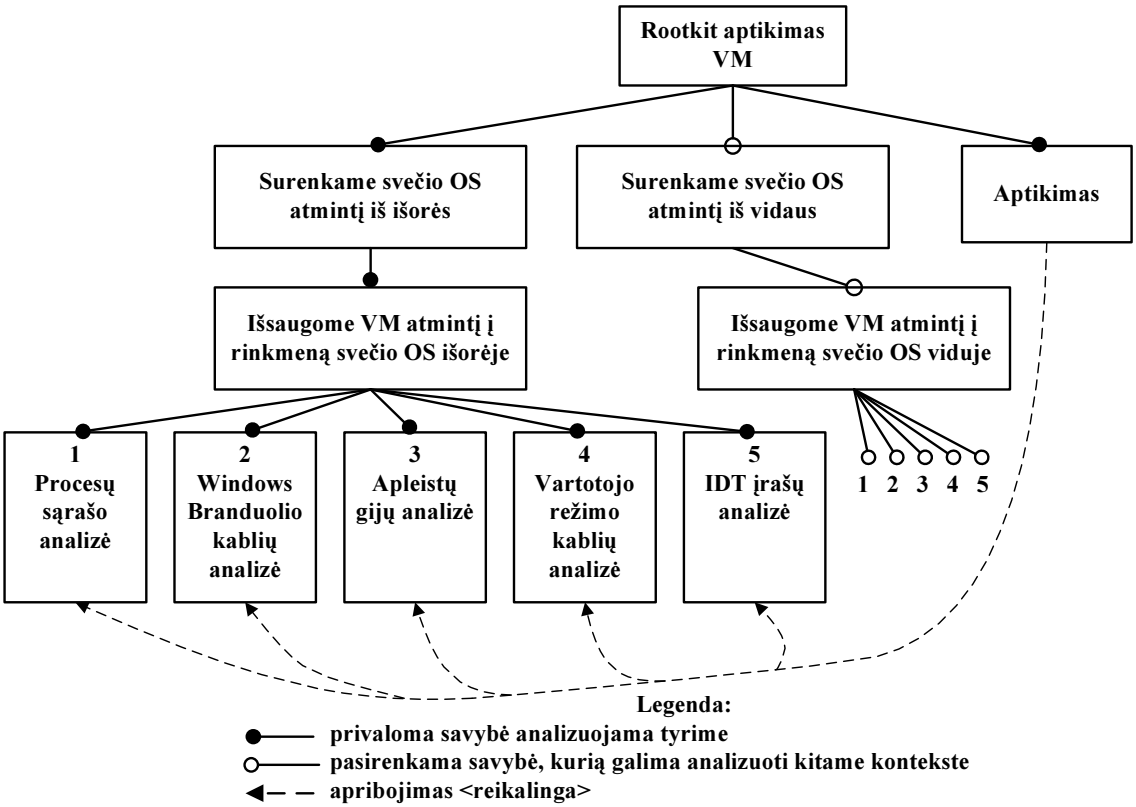
Šią metodiką geriausiai aprašo medžio tipo savybių diagrama. Diagramos mazgai (stačiakampiai) nusako savybes (juodi rutuliukai nusako būtinąsias, balti pasirenkamąsias).

Šakos tarp mazgų nusako ryšius tarp savybių. Šiuo atveju turime dviejų tipų ryšius. „tėvas-vaikas”- aprašyti ryšiams medyje ir „ryšių” nusakyti aprašyti ryšiams tarp galutinių taškų.

11 pav. nurodoma metodikos nagrinėjama sritis ir priėjimo prie eksperimentui reikalingų duomenų realizacija. Buvo pasirinktas statinis virtualių mašinų analizės metodas , kuris remiasi virtualių mašinų atminties vaizdų (image) analizavimu. Nors pagal 11 pav. savybių diagramą tokia analizė laikoma statine, ją galima iš dalies paversti dinamiška, panaudojant momentines virtualių mašinų kopijas ir tokiu būdu išvengti virtualių mašinų darbo trikdymo. Pilnas analizės dinamiškumas būtų įmanomas turit betarpišką priėjimą prie virtualių mašinų darbinės atminties.

Priėjimo prie duomenų realizacijos šakoje yra naudojamas virtualios mašinos struktūros atkūrimas, o būtent - virtualios mašinos darbinės atminties struktūros atkūrimas iš darbinės atminties momentinės kopijos.

Toliau nagrinėjama „rootkit“ tipo piktavališkos programinės įrangos aptikimo virtualių mašinų aplinkoje savybių diagrama.



12 pav. piktavališkos programinės įrangos aptikimo metodikos savybių diagrama

12 pav. savybių diagramoje matome darbo metodikos sudarymui ir tyrimui naudojamus ir analizuojamus piktavališkos programinės įrangos aptikimo metodus.

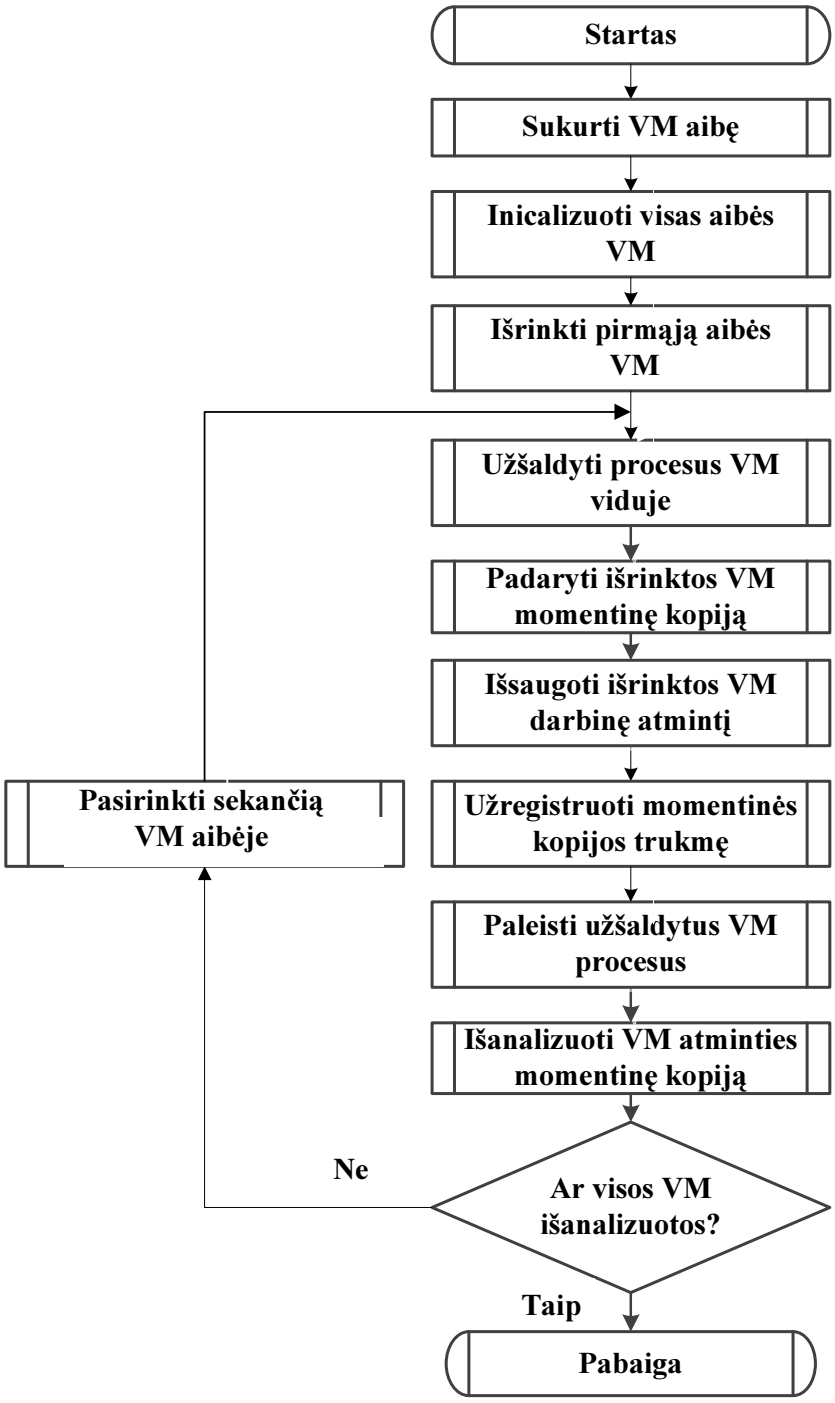
Pagrindinė metodų kryptis – stebėti virtualizuotą operacinę sistemą iš išorės, tai realizuojama hipervizoriaus priemonėmis išsaugant operacinės sistemos darbinę atmintį – RAM. Žemesniuose diagramos lygiuose nusakoma kokie analizės metodai naudojami – procesų sąrašo analizė, Windows branduolio sisteminių lentelių įrašų analizė, apleistų gijų analizė, vartotojo režimo sisteminių įrašų analizė ar pertraukimų aprašymo lentelės analizė.

3.1. Klastingos programinės įrangos aptikimo metodikos tyrimo strategijos algoritmas

Klastingos piktavališkos programinės įrangos aptikimui virtualių mašinų aplinkoje naudojamas toks algoritmas:

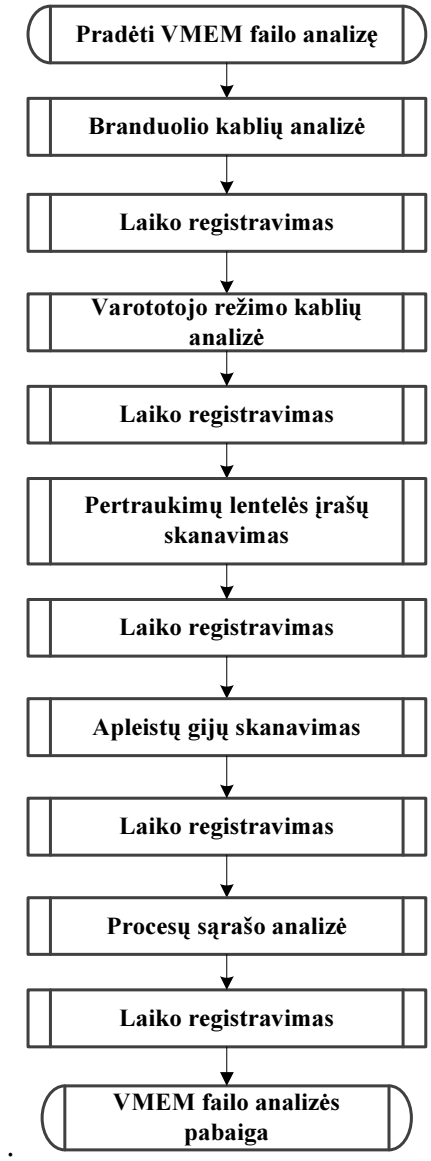
1) Sukuriama virtualių mašinų aibė (tai gali būti ir jau sukurtų, dirbančių virtualių mašinų aibė)
2) Jeigu virtualių mašinų aibė yra naujai sukurta ji yra inicializuojama (inicializuojamos visos virtualios mašinos esančios grupėje). Inicializacijos metu paleidžiama operacinė sistema, taikomosios ir pagalbinės programos , o taip pat tiriamoji piktavališka programinė įranga
3) Išrenkama pirmoji aibės virtuali mašina . Išrinkimas atliekamas operatoriaus nuožiūra.
4) Naudojant pagalbinę programinę įrangą (šio darbo atveju Flypaper) užšaldomi visi procesai dirbantys virtualios mašinos viduje.
5) Sukuriama virtualios mašinos momentinė kopija
6) Išsaugoma momentinės kopijos darbinė atmintis, būtent ji yra reikalinga analizei
7) Užšaldyti virtualios mašinos procesai paleidžiami toliau dirbti
8) Analizuojama virtualios mašinos darbinės atminties momentinė kopija
9) Atlikus atminties momentinės kopijos analizę pasirenkama sekanti aibėje esanti virtuali mašina ir žingsniai 4 – 8 pakartojami
10) Kai išanalizuotos visos virtualios mašinos, algoritmas baigiamas

Algoritmo blokinė diagrama pateikta 13 paveiksle.



13pav. - Piktavališkos programinės įrangos virtualioje aplinkoje aptikimo algoritmas

Detaliau panagrinėsiu analizės žingsnio algoritmą. Kai sukuriama virtualios mašinos momentinė kopija ir išsaugomas darbinės atminties atvaizdas , pradedama atminties atvaizdo analizė



14 pav. Virtualios mašinos darbinės atminties analizės algoritmas

Analizės metu atliekami šie žingsniai:

1) Inicijuojama analizė – paleidžiamas Volatility karkaso scenarijus.
2) Atliekama branduolio sisteminių įrašų analizė, kurios metu tikrinami branduolio sisteminių įrašų lentelės įrašai
3) Užregistruojamas analizei sugaištas laikas
4) Atliekama vartotojo režimo sisteminių įrašų analizė, kurios metu tikrinama vartotojo režimo sisteminių įrašų lentelė
5) Užregistruojamas analizei sugaištas laikas
6) Atliekama pertraukimų aprašymo lentelės įrašų patikra, kurios metu ieškoma įtartinų įrašų

7) Užregistruojamas analizei sugaištas laikas
8) Atliekamas apleistų gijų analizė, kurios metu darbinėje atmintyje ieškoma gijų nesusietų su žinomais procesais
9) Užregistruojamas analizei sugaištas laikas
10) Atliekama procesų sąrašo analizė, kurios metu iš darbinės atminties momentinės kopijos išrenkamas procesų sąrašas
11) Užregistruojamas analizei sugaištas laikas

Šiame žingsnyje užregistruotos laiko sąnaudos yra naudojamos eksperimento rezultatų analizėje. Blokinė algoritmo diagrama pateikiama 14 pav.

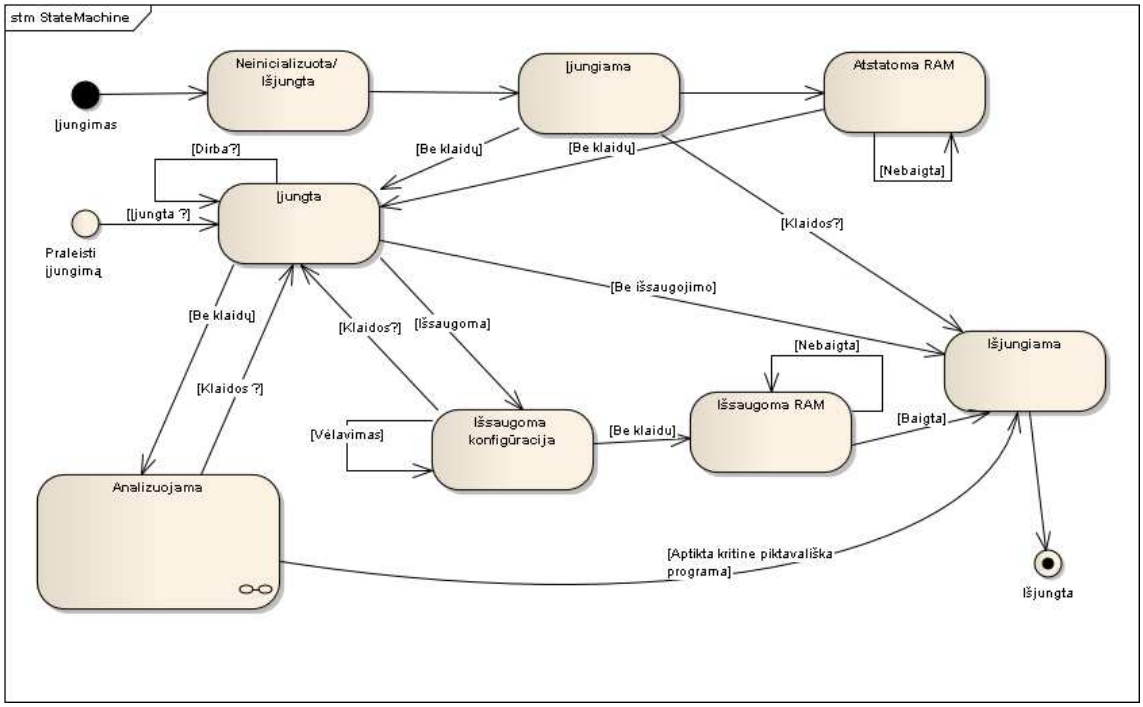
Tam, kad pagreitinti virtualių mašinų inicializavimo ir analizės procesus, buvo panaudoti komandų scenarijai. Jie pateikiamas virtualios aplinkos konfigūracijos, naudotos tyrime, apžvalgos skyriuje.

Virtualių mašinų inicializavimo pagalbinis scenarijus pavaizduotas– 18 pav.

Atminties atvaizdų analizės automatizavimo scenarijus pavaizduotas– 19 pav.

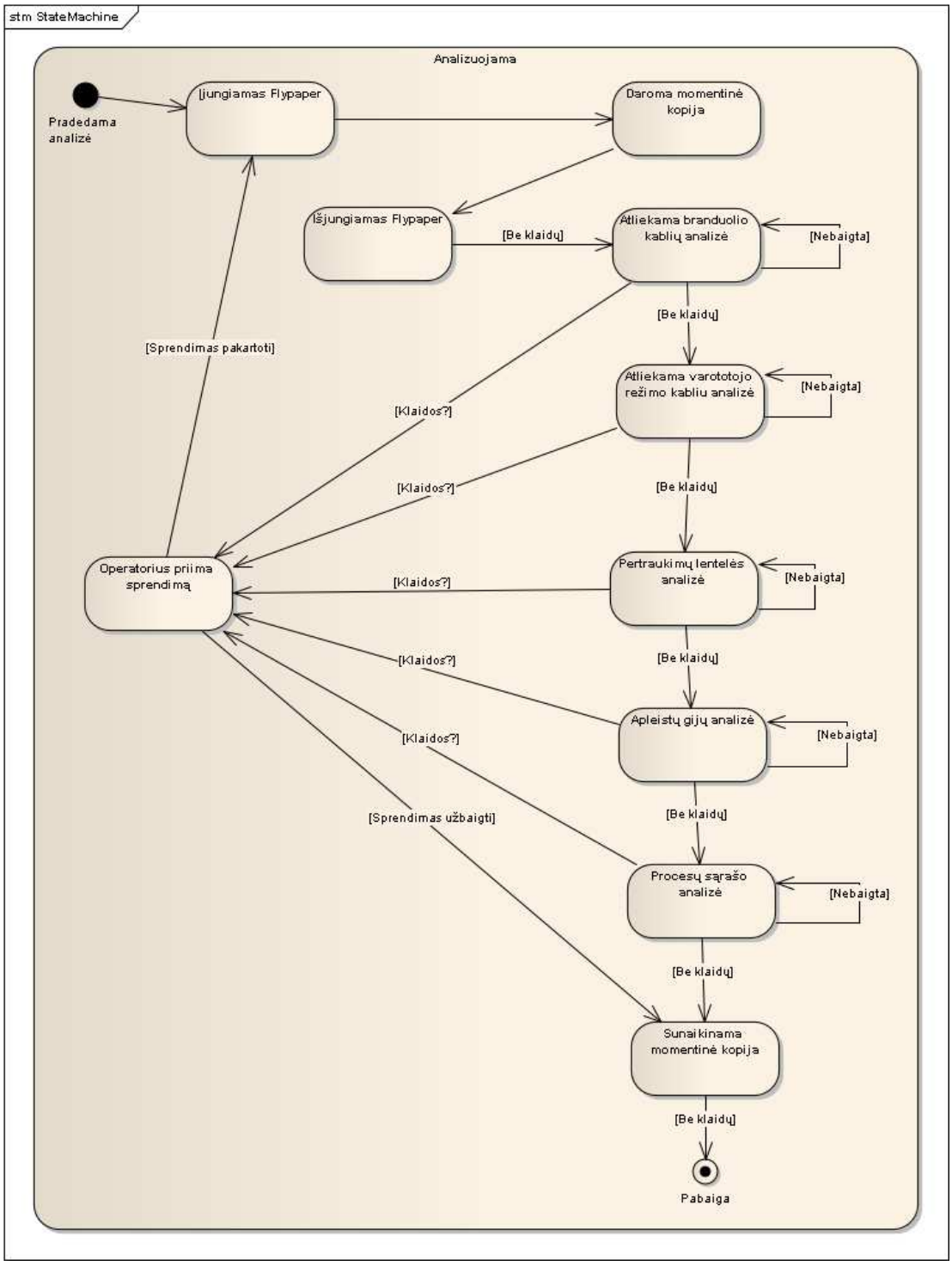
Būsenų diagramos

Būsenos kurias analizės metu patiria virtuali mašina aprašytos UML State machine būsenų diagramomis.



15 pav. Virtualios mašinos būsenų diagrama, piktavališkos programinės įrangos paieškos metu.

Virtualios mašinos būsenų diagrama, piktavališkos programinės įrangos paieškos metu pavaizduota 15 paveiksle, o detali būsenos „Analizuojama“ diagrama pateikta 16 paveiksle.



16 pav. – Virtualios mašinos būsenos - „Analizuojama“, detali būsenų diagrama

Reiktų atkreipti dėmesį jog būsenos „Analizuojama“ , klaidų atveju sprendimą turi priimti operatorius. Sprendimo priėmimo algoritmas ir automatizavimas nepatenka į šio darbo rėmus. Tačiau gali būti naudojamas darbo tęstinumui.

3.2. Klastingos programinės įrangos aptikimo metodikos įrankiai

Metodikoje naudojami įrankiai:

- ✓ **Volatility karkasas** – Volatility karkasas tai atviro kodo pagrindu, Python kalba sukurtų, ir pagal GNU (General Public License) licenciją platinamų įrankių rinkinys, skirtas išsaugotos darbinės (RAM) atminties analizei ir joje randamų skaitmeninių artefaktų *ekstrahavimui*. Išsaugojimo metodai yra visiškai nepriklausomi nuo tiriamos platformos ir suteikia galimybę nepaprastai detaliai ištirti darbinės atminties turinį. [30].
- ✓ **Volatility karkaso įskiepiai** – oficialiai prieinami Volatility karkaso įskiepiai [31].
- ✓ **Python** – objektinė, interaktyvi interpretuojamoj programavimo kalba [32].
- ✓ **MinGW** – GNU Compiler Collection (GCC) ir GNU Binutils įrankių rinkinys pritaikytas programavimui Microsoft Windows aplinkoje. Mūsų atveju buvo naudojamas Volatility karkaso įskiepių programavimui [33].
- ✓ **Flypaper.exe** – HBGary Flypaper taikomoji programa pakraunama kaip įrenginio tvarkyklė ir blokuoja visus bandymus sustabdyti procesą, užbaigti giją ar ištrinti atminties segmentą. Visi piktavališkos programinės įrangos komponentai užrakinami darbinėje fizinėje atmintyje. Visi programų vykdymo etapai yra išsaugomi tad galima sekti kiekvieną proceso žingsnį. HBGary Flypaper nekomerciniam naudojimui yra nemokamas[34].
- ✓ **VMware Workstation v7** – Kompiuterinėms darbo vietoms skirtas virtualizacijos sprendimas, dirbanti II tipo hipervizoriaus principu [35].

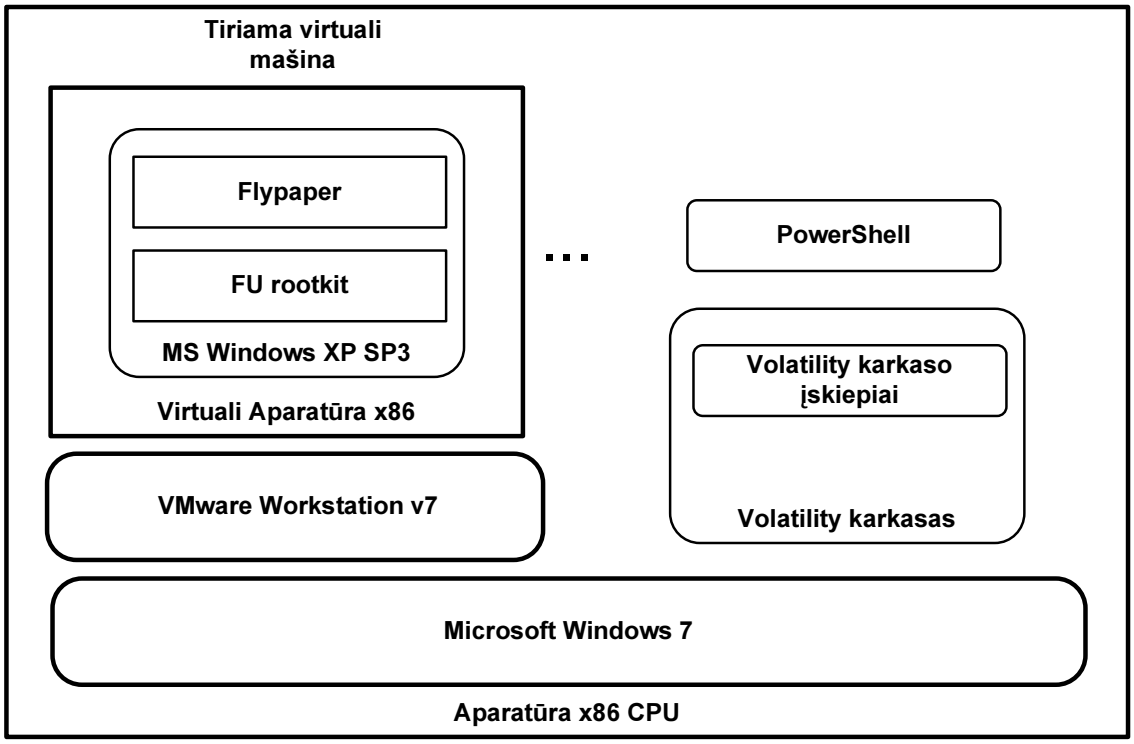
3.3. Virtualios aplinkos konfigūracija naudota metodikos tyrimui atlikti.

Virtualios aplinkos konfigūracija buvo realizuota II tipo hipervizoriaus pagrindu. Eksperimento darbinę aplinką sudaro:

- ✓ „Šeimininko“ operacinė sistema – MS Windows 7

- ✓ „Svečio“ operacinė sistema – MS Windows XP SP3
- ✓ VMware Workstation v7– darbo stotims skirtas virtualizavimo sprendimas[35]. Jo pagalba buvo kuriamos virtualios mašinos (VM) ir daromos jų momentinės kopijos (Snapshot).
- ✓ PowerShell – Windows komandų aplinka, skirta užduočių automatizavimui
- ✓ Volatility karkasas ir jo įskiepiai
- ✓ FU rootkit – klastinga piktavališka programinė įranga
- ✓ HBGary Flypaper – pagalbinė programa

Darbinės aplinkos schema pateikiama 17 pav.



17 pav. – eksperimento darbinės aplinkos schema

Detali, eksperimente naudotos, virtualios mašinos konfigūracija pateikiama 18pav. Norint kuo tiksliau atkartoti eksperimentą rekomenduojama naudoti paveiksle pateiktą konfigūraciją.

```
.encoding = "windows-1252"
config.version = "8"
virtualHW.version = "7"
```

```
maxvcpus = "4"
scsi0.present = "TRUE"
memsize = "512"
ide0:0.present = "TRUE"
ide0:0.fileName = "Windows XP Professional SP3-cl3-000012.vmdk"
ide1:0.present = "TRUE"
ide1:0.autodetect = "TRUE"
ide1:0.deviceType = "cdrom-image"
ethernet0.present = "TRUE"
ethernet0.connectionType = "nat"
ethernet0.wakeOnPcktRcv = "FALSE"
ethernet0.addressType = "generated"
ehci.present = "TRUE"
mks.enable3d = "TRUE"
pciBridge0.present = "TRUE"
pciBridge4.present = "TRUE"
pciBridge4.virtualDev = "pcieRootPort"
pciBridge4.functions = "8"
pciBridge5.present = "TRUE"
pciBridge5.virtualDev = "pcieRootPort"
pciBridge5.functions = "8"
pciBridge6.present = "TRUE"
pciBridge6.virtualDev = "pcieRootPort"
pciBridge6.functions = "8"
pciBridge7.present = "TRUE"
pciBridge7.virtualDev = "pcieRootPort"
pciBridge7.functions = "8"
vmci0.present = "TRUE"
roamingVM.exitBehavior = "go"
displayName = "Lab01"
guestOS = "winxppro"
nvram = "Lab01.nvram"
virtualHW.productCompatibility = "hosted"
extendedConfigFile = "Lab01.vmxf"
```

```
ethernet0.generatedAddress = "00:0c:29:33:fd:75"
uuid.location = "56 4d ac 1a 61 22 80 b0-c6 cf 71 f7 86 33 fd 75"
uuid.bios = "56 4d ac 1a 61 22 80 b0-c6 cf 71 f7 86 33 fd 75"
cleanShutdown = "TRUE"
replay.supported = "FALSE"
replay.filename = ""
ide0:0.redo = ""
pciBridge0.pciSlotNumber = "17"
pciBridge4.pciSlotNumber = "21"
pciBridge5.pciSlotNumber = "22"
pciBridge6.pciSlotNumber = "23"
pciBridge7.pciSlotNumber = "24"
scsi0.pciSlotNumber = "16"
ethernet0.pciSlotNumber = "33"
ehci.pciSlotNumber = "35"
vmci0.pciSlotNumber = "36"
vmotion.checkpointFBSize = "134217728"
usb:0.present = "TRUE"
usb:1.present = "TRUE"
ethernet0.generatedAddressOffset = "0"
vmci0.id = "-334199147"
usb:1.deviceType = "hub"
usb:0.deviceType = "mouse"
ide1:0.fileName = ""
tools.remindInstall = "FALSE"
vc.uuid = ""
inVMTeam = "FALSE"
isolation.tools.hgfs.disable = "FALSE"
sharedFolder0.present = "TRUE"
sharedFolder0.enabled = "TRUE"
sharedFolder0.readAccess = "TRUE"
sharedFolder0.writeAccess = "TRUE"
sharedFolder0.hostPath = "C:\Studies\wormhole"
sharedFolder0.guestName = "wormhole"
```

```
sharedFolder0.expiration = "never"
sharedFolder.maxNum = "1"
hgfs.mapRootShare = "TRUE"
tools.syncTime = "FALSE"
ide1:0.startConnected = "FALSE"
unity.wasCapable = "FALSE"
monitor.virtual_mmu = "automatic"
monitor.virtual_exec = "automatic"
checkpoint.vmState.readOnly = "FALSE"
checkpoint.vmState = ""
policy.vm.managedVMTemplate = "FALSE"
policy.vm.managedVM = "FALSE"
usb.present = "FALSE"
sound.present = "FALSE"
floppy0.present = "FALSE"
```

18 pav. detali eksperimente naudotos virtualios mašinos konfigūracija

Tam, kad automatizuoti virtualių mašinų paruošimą eksperimentui, FU Rootkit klastingos piktavališkos programinės įrangos įdiegimui, buvo naudojamas komandų scenarijus (18 pav.) Jis buvo paleidžiamas kiekvienos virtualios mašinos inicializavimo metu.

```
@echo off
REM Idiegia fu root kit
REM nukopijuoti rinkmenas
SET install="%systemroot%\system32\instdriver.exe"
:copy
copy FU_Rootkit\FU_Rootkit\EXE\fu.exe%systemroot%\system32
\wsctntty.exe
copy FU_Rootkit\FU_Rootkit\EXE\msdirectx.sys %systemroot%
\system32\msdirectx.sys
copy InstDriver\InstDriver.exe %systemroot%\system32
\instdriver.exe
:Install
%install% -Install msdirectx %systemroot%\system32
\msdirectx.sys
%install% -Start msdirectx
Echo Idiegta!
```

19 pav. Piktavališkos programinės įrangos įdiegimo scenarijus

Išsaugotiems virtualių mašinų atminties pavyzdžiams analizuoti buvo naudojamas PowerShell komandų scenarijus - 19pav. Jo pagalba buvo paleidžiamos Volatility karkaso komandos ir registruojami laiko sąnaudų duomenys.

```
Clear-Host
#nustatomi kintamieji
#nustatomas kelias kur ieškoti atminties atvaizdu
$vmemdir = "C:\Studies\VM\vmems"
#nustatomas kelias laikiniems failams
$temp = "-d C:\temp"
#sudaromas tik atminties atvaizdu sąrašas
$list = @(Get-ChildItem -Path $vmemdir -name -Include *.vmem)
#apdorojamas atvaizdu sąrašas
foreach ($i in $list)
{measure-command {python volatility.py pslist -f  "$vmemdir\$i"}}
foreach ($i in $list)
{measure-command {python volatility.py orphan_threads  -f  "$vmemdir\$i"}}
foreach ($i in $list)
{measure-command {python volatility.py idt_entries  -f  "$vmemdir\$i"}}
foreach ($i in $list)
{measure-command {python volatility.py usermode_hooks -f "$vmemdir\$i" $temp}}
foreach ($i in $list)
{measure-command {python volatility.py kernel_hooks -f  "$vmemdir\$i" $temp}}
```

20 pav. PowerShell komandų scenarijus (script) naudojamas analizei automatizuoti

4. Eksperimentinis klastingos programinės įrangos aptikimo metodikos tyrimas

Analizuojant prieinamą informaciją apie virtualizuotų operacinių saugos sprendimus pastebėjau jog dėl uždaro kodo buvo atlikta tik dalis tyrimų su VMware sprendimais [1].
Šiuo metu ribotam saugos sprendimų kūrėjų ratui yra prieinama VMware API – VMSafe, kuri leidžia realizuoti funkcijas, kurių realizacija dar neseniai buvo neįmanomą, tačiau VMSafe kodas yra uždaras ir jo šiame darbe panaudoti nepavyko. Todėl buvo ieškoma tokia sistemos informacijos paėmimo sritis, kuri būtų nepriklausoma nuo kodo uždarumo.

Buvo atrasta jog VM momentinių kopijų darbinės atminties rinkmenos, nėra apribotos uždaro kodo ir atitinka atminties atvaizdavimo rinkmenose standartus.

Todėl eksperimentinėje dalyje susitelksiu ties VM atminties vaizdo turinio analize (lentelėje pažymėta ☑.

Lentelė 3. Eksperimentinės dalies pasirinkimas

VMM stebėjimo lygis	**Pilna virtualizacija**		**Paravirtualizacija**	
	VMware	**QEMU**	**Xen**	**UML**
VM disko vaizdas (raw disk image)	X	X	X	X
VM Atminties vaizdas (raw memory image)	☑	X	X	X
VM aparatinės būklės (valdymo registrai)	X	X	X	X
Su VM susiję žemo lygio įvykiai (pertraukimai, spąstai)	X	X	X	X

4.1. Eksperimento eiga

Buvo naudojama penkių pakopų metodika:

- ✓ Aplinkos inicializacija

- ✓ Momentinių kopijų sukūrimas
- ✓ Darbinės atminties pavyzdžių analizė
- ✓ Kiekvienos atliktos analizės rezultatų registravimas
- ✓ Surinktų rezultatų grafinė analizė

Aplinkos inicializacijos pakopos veiksmai:

- ✓ Pirmiausia sukuriama etaloninė virtuali mašina
- ✓ Įdiegiama etaloninės virtualios mašinos operacinė sistema
- ✓ Naudojant virtualių mašinų klonavimo (Clone) funkcionalumą, sukuriamos likusios eksperimente dalyvausiančios virtualios mašinos.
- ✓ Aibės narių darbinės atminties dydis svyruoja nuo 128MB iki 2048MB, augimo žingsnis 128MB.

Sekantis etapas – momentinių kopijų sukūrimas. Verta paminėti jog prieš darant momentinę kopiją, siekiant užtikrinti kuo efektyviau aptikti piktavališką programinę įrangą, atliekamas visų procesų užfiksavimas atmintyje. Tą mums leidžia padaryti HB Gary Flypaper programa, užblokuojanti:

- ✓ procesų užbaigimo funkcijas
- ✓ gijų užbaigimo funkcijas
- ✓ bandymus ištrinti atminties segmentą.

Detaliau panagrinėsime momentinių kopijų sudarymo veiksmus:

- ✓ Paruoštai (su paleista Flypaper piktavališka programine įranga)virtualiai mašinai rankiniu būdu buvo paleidžiama momentinės kopijos kūrimo komanda.
- ✓ Momentinės kopijos kūrimo komandai pasibaigus buvo perskaitomas virtualios mašinos žurnalas ir užregistruojamas laikas reikalingas sukurti momentinei kopijai.
- ✓ Išsaugojama virtualios mašinos darbinės atminties momentinės kopijos rinkmena – .VMEM.
- ✓ Virtualioje mašinoje išjungiama flypaper.exe ir paleidžiama operacinės sistemos išjungimo komanda.

- ✓ Šioje pakopoje surinktos darbinės atminties momentinės kopijos perkeliamos į numatytą Volatility karkasui darbinį katalogą.

Darbinės atminties pavyzdžių analizės etapas:

- ✓ Iš PowerShell aplinkos paleidžiamas Volatility karkaso komandas automatizuojantis scenarijus ir atliekama sukauptų atminties momentinių kopijų rinkmenų analizė
- ✓ Naudojant PowerShell žurnalą ir MS Excel užregistruojami kiekvienos analizės metu sugaišto laiko rezultatai

Surinktų rezultatų analizės etapas:

- ✓ Naudojant MS Excel surinkti duomenys atvaizduojami grafiškai
- ✓ Pateikiamos eksperimento išvados

4.2. Virtualios aplinkos konfigūracija naudota metodikos tyrimui atlikti

Virtualios mašinos konfigūracija naudota eksperimento metu aprašoma 3.3 skyriuje.

Kiekviena virtuali mašina inicializuojama šiuo būdu:

- ✓ Įdiegiama gerai žinoma klastinga piktavališka programinė įranga – FU rootkit. [36]
- ✓ Įdiegiamos tinklo paslaugos (tokios kaip - tftpd, ftpd, syslogd), komunikavimui su virtualia mašina ir piktavališkų tinklo procesų imitavimui, jos slepiamos naudojant FU rootkit.
- ✓ Įdiegiama HB Gary Flypaper.exe programa skirta procesų operacinėje sistemoje *užšaldymui*.

Pagrindinė (host) operacinė sistema paruošiama sekančiai:

- ✓ Įdiegiama Python v2.6 vykdymo aplinkos versija, nes Volatility karkasas yra parašytas būtent šia kalba
- ✓ Įdiegiamas Volatility v1.3 karkasas
- ✓ Įdiegiami Volatility karkaso įskiepiai (plug-ins)

- ✓ Jei nėra operacinėje sistemoje įdiegiama Microsoft PowerShell aplinka, ji reikalinga komandų automatizavimui ir turi funkcijas komandų vykdymo laikui matuoti.

4.3. Eksperimento atlikimas

- ✓ Eksperimentas buvo atliekamas pagal - klastingos programinės įrangos aptikimo metodikos tyrimo strategijos ir Virtualios mašinos darbinės atminties analizės algoritmus.
- ✓ Algoritmų žingsniai buvo įgyvendinami vadovaujantis metodikos etapuose išdėstytais nurodymais.
- ✓ Norint gauti kuo objektyvesnius rezultatus eksperimento metu nebuvo leidžiamos jokios papildomos taikomosios programos.

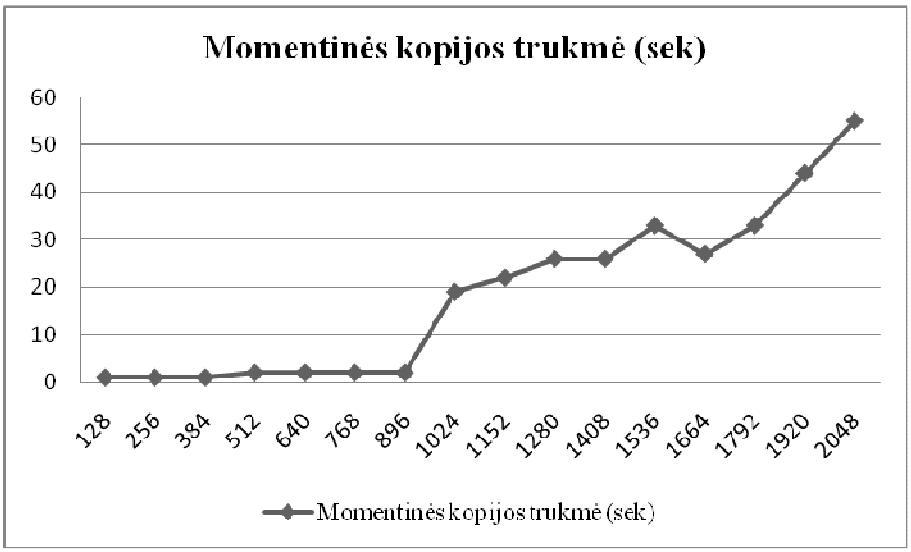
4.4. Eksperimento rezultatai

Lentelė 4. Eksperimento rezultatai

VM atminties dydis MB	Momentinės kopijos trukmės	Apleistų gijų analizės	Procesų sąrašo analizės	Branduolio sisteminių įrašų analizės	Vartotojo režimo sisteminių įrašų analizės	IDT įrašų analizės	Laiko suma s
128	1	3,88	0,14	2,65	127,10	0,77	135,55
256	1	1,12	0,14	2,59	119,72	0,33	124,91
384	1	4,63	0,14	2,57	116,88	0,36	125,59
512	2	6,17	0,14	2,58	116,21	0,42	127,52
640	2	6,70	0,16	2,56	100,38	0,49	112,28
768	2	7,79	0,14	2,58	109,60	0,38	122,50
896	2	8,79	0,14	2,53	108,99	0,50	122,96
1024	19	22,34	0,14	2,56	101,38	0,54	145,97
1152	22	25,93	0,14	2,60	97,35	0,79	148,80
1280	26	27,86	0,14	2,59	108,73	0,56	165,88
1408	26	28,58	0,14	2,53	113,36	0,52	171,14
1536	33	30,72	0,14	2,60	116,85	0,53	183,83

1664	27	33,05	0,14	2,58	121,85	0,56	185,19
1792	33	39,82	0,14	2,55	108,97	0,48	184,96
1920	44	43,66	0,15	2,59	113,93	0,55	204,87
2048	55	46,58	0,14	2,58	117,96	0,40	222,67

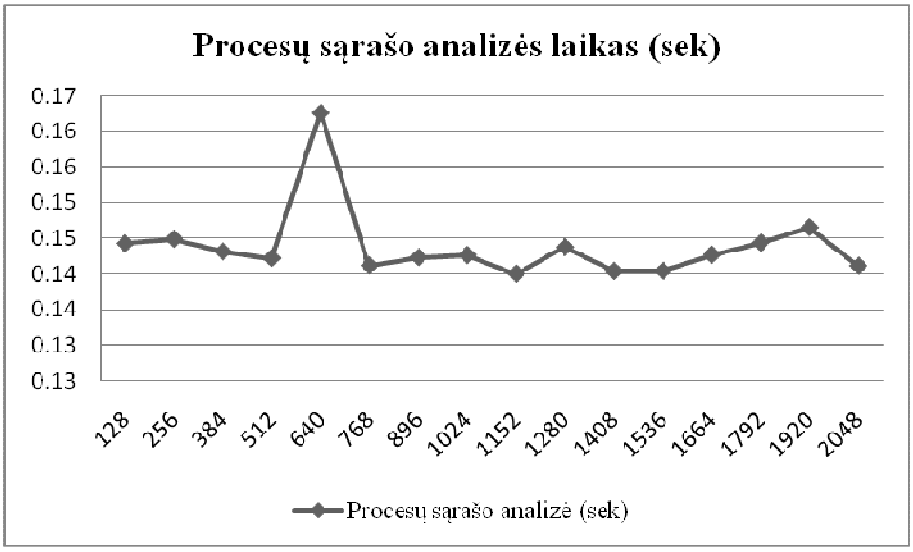
Lentelėje 4 ir 20 paveiksle matome jog momentinių kopijų trukmė tiesiogiai įtakojama virtualios mašinos atminties dydžio. Reikia pastebėti jog darant virtualių mašinų, kurių atmintis iki 1GB momentines kopijas momentinių kopijų vykdymo trukmė gerokai trumpesnė nei virš 1GB. Taip yra todėl, kad kai hipervizoriaus konfigūracijoje įjungtas parametras „Take snapshots in the background" (mainMem.partialLazySave = TRUE), <1GB RAM VM momentinės kopijos metu vartotojui paliekama galimybė nenutrūkstamai dirbti su VM. Vartotojui iškart pranešama, kad momentinė kopija padaryta, o antrame plane duomenys iš lėto surašomi į diską. VM kurių RAM >1GB, minėtas funkcionalumas automatiškai išjungiamas. Tokiu atveju momentinės kopijos kūrimo metu, kol visi duomenys surašomi į diską, dirbti su VM negalima.



20 pav. Momentinės kopijos trukmė

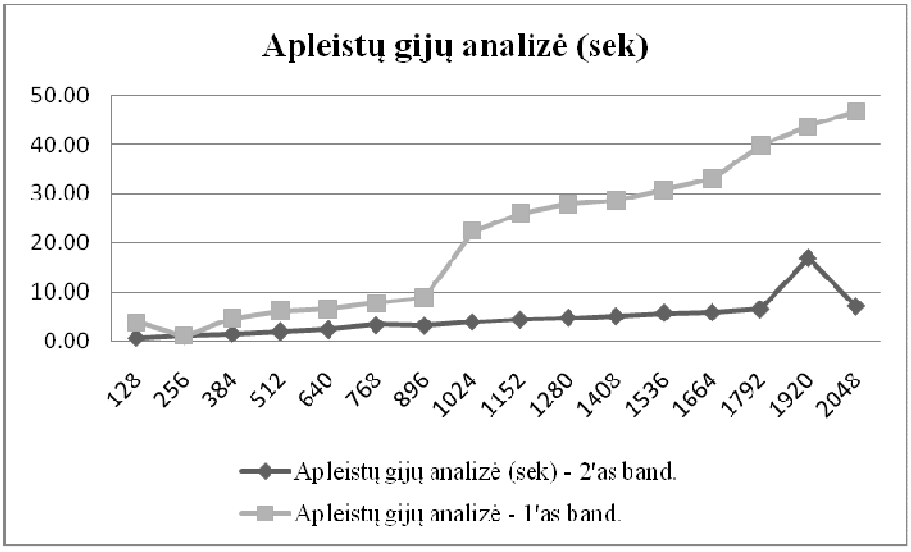
Toliau atliekama procesų sąrašo analizė. Jos metu kiekviename darbinės atminties pavyzdyje ieškoma tuo metu operacinėje sistemoje veikusių procesų. Kai randami ir sunumeruojami visi procesai vartotojui atliekančiam analizę pateikiamas procesų sąrašas. Analizės trukmė nuo darbinės atminties dydžio nepriklauso, nes informacija apie operacinės sistemos procesus laikoma, gerai žinomame atminties segmente. Laiko pikas, kuris

pastebimas analizuojant pavyzdžius nuo 512MB iki 768MB, yra atsiradęs dėl tuo metu padidėjusio „šeimininko“ operacinės sistemos apkrovimo. Nors siekiant užtikrinti duomenų tikslumą „šeimininko“ operacinėje sistemoje nebuvo leidžiami jokie papildomi procesai ar taikomosios programos, visiško analizės proceso izoliacijos sukurti nepavyko. Piko skirtumas nuo kitų rezultatų yra 0,3sek, bendrai analizės trukmei tai didelės įtakos neturi.



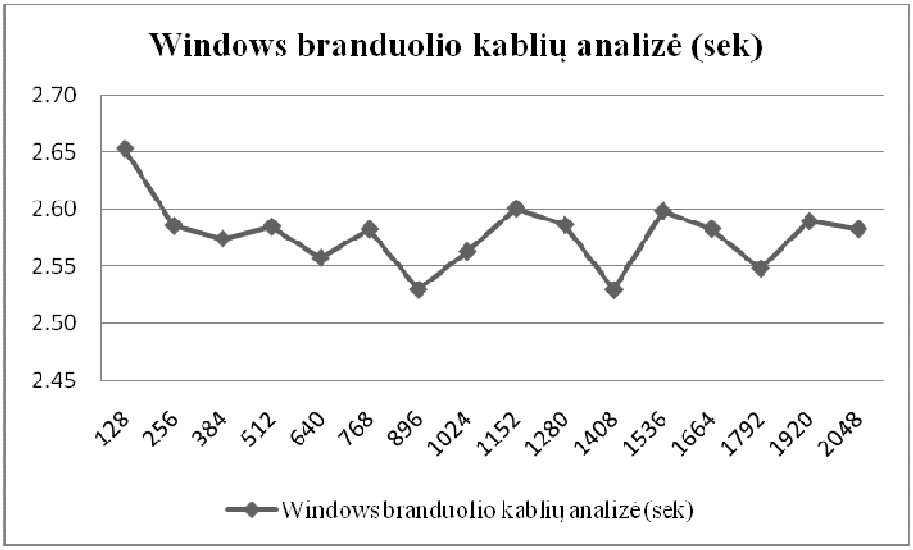
21 pav. Procesų sąrašo analizės trukmė

Toliau buvo atliekama apleistų gijų, naudojamų paslėpti piktavališką programinę įrangą analizė (Pav.22). Šią analizę eksperimento metu kartojau du kartus. Pakartotinis bandymas buvo atliekamas tam, kad įvertinti „šeimininko“ operacinės sistemos spartinančiosios atminties poveikį analizei. Pirmojo bandymo metu pastebima analizės trukmės priklausomybė nuo darbinės atminties dydžio. Antrojo bandymo metu, ypač su mažesniais darbinės atminties pavyzdžiais, pastebimas spartinančiosios atminties efektas.



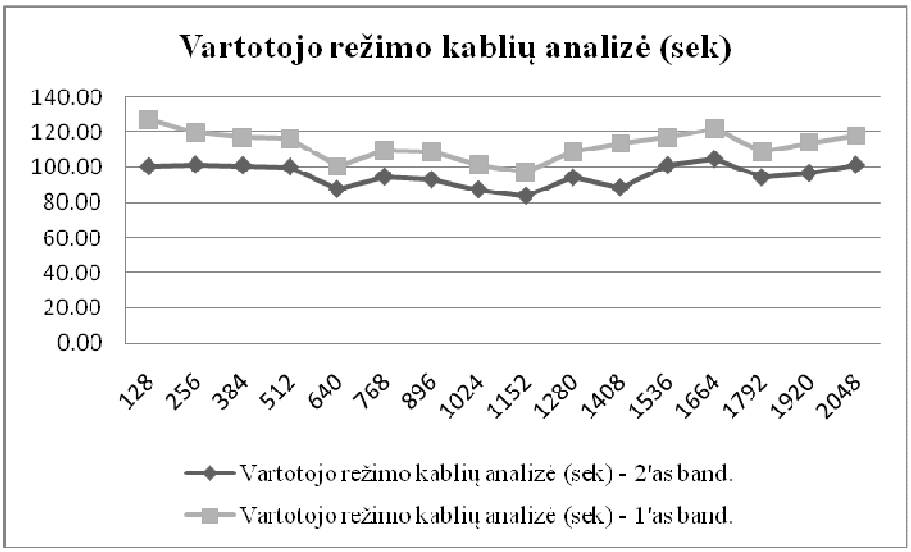
22 pav. Apleistų gijų analizės trukmė

Windows branduolio sisteminių įrašų analizės (23 pav.) metu pastebimi grafiko svyravimai, kadangi nepastebima priklausomybė nuo virtualios mašinos darbinės atminties dydžio. Kol neatliksime papildomų eksperimentų tai galima paaiškinti tik kaip apkrovimo svyravimais „šeimininko“ operacinėje sistemoje. Bendros analizės trukmės šie svyravimai beveik neįtakoja.



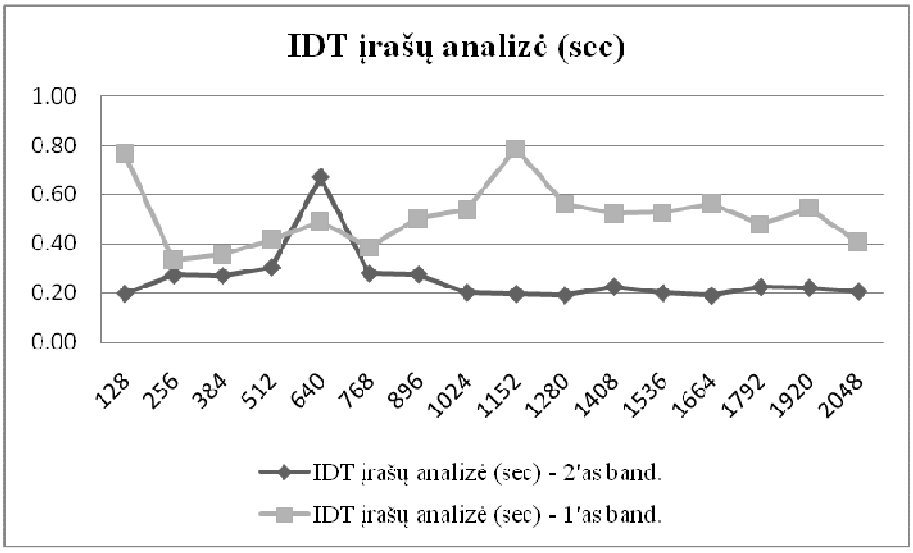
23 pav. Windows branduolio sisteminių įrašų analizės trukmė

Vartotojo režimo sisteminių įrašų analizės metu (24 pav.) taip pat kaip apleistų gijų analizės metu pastebimas „šeimininko“ operacinės sistemos spartinančios atminties efektas. Tam kad įvertinti spartinančios atminties įtaką eksperimentas buvo kartojamas keletą kartų.



24 pav. Vartotojo režimo sisteminių įrašų analizės trukmė

Atliekant pertraukimų lentelės analizę pastebimas tiek „šeimininko“ operacinės sistemos spartinimo efektas tiek rezultatų netolygumas dėl kintančio sistemos apkrovimo.



25 pav. Pertraukimų aprašymo lentelės analizės trukmė

Priklausomybė nuo virtualios mašinos atminties dydžio nepastebima.

5. Išvados

Įvertinus šio magistro darbo metu surinktą patirtį ir eksperimento keliu gautus duomenis, galima daryti šias išvadas:

- ✓ Magistro darbo rezultatai buvo pristatyti tarptautinėje konferencijoje „Elektronika 2010“. Paruoštas publikacijai straipsnis: „Rootkit detection experiment within a virtual environment“. Konferencijos dalyvio diplomas ir straipsnio tekstas pateikti prieduose.
- ✓ Klastinga piktavališka programinė įranga pastoviai tobulėja, naujausi pasiekimai šioje srityje naudoja virtualizaciją pėdsakams paslėpti. Tai galėtų būti šio darbo tęstinumas.
- ✓ Nepaliaujamai tobulinami piktavališkos programinės įrangos aptikimo metodai, virtualizacija galime naudoti kaip vieną iš technologijų leidžia įgyvendinti naujus metodus.
- ✓ Informacijos saugos problemos aktualios ir virtualizuotose aplinkose. Atsiranda tiek organizacinių tiek technologinių informacijos saugos iššūkių.
- ✓ I ir II tipų klastingos piktavališkos programinės įrangos aptikimą įmanoma patobulinti naudojantis virtualizacijos sprendimais.
- ✓ Piktavališkos programinės įrangos paieškos trukmė, kai naudojamas virtualių mašinų darbinės atminties atvaizdai, tiesiogiai priklauso nuo darbinės atminties dydžio.
- ✓ Eksperimento rezultatus dažnai įtakoja „šeimininko“ operacinės sistemos apkrovimo svyravimai ir spartinančiosios atminties efektas.
- ✓ Piktavališkos programinės įrangos paiešką virtualiose mašinose galima atlikti pasitelkiant standartinių virtualizacijos sprendimų suteikiamas priemones, šiuo atveju momentines kopijas
- ✓ Siekiant pagreitinti piktavališkos programinės įrangos paiešką virtualiose mašinose ją įmanoma automatizuoti
- ✓ Norint, kad aptikimo rezultatai būtų tikslūs reikia pritaikyti visus įmanomus analizės tipus, nes piktavališka programinė įranga naudoja visus įmanomus maskavimosi metodus

6. Naudota literatūra

1. BIANCUZZI F. Windows rootkits come of age http://www.securityfocus.com/columnists/358/1 (tikrinta 2010.05)
2. BLUNDEN, B. The Rootkit Arsenal: Escape and Evasion in the Dark Corners of the System. 2009. 908p. ISBN-13: 978-1598220612
3. BORDER C. The Development and Deployment of a Multi-User, Remote Access Virtualization System for Networking, Security, and System Administration Classes. ACM SIGCSE Bulletin. Volume 39 , Issue 1 (March 2007)
4. BOWMAN M., BROWN H., PITT P. An Undergraduate Rootkit Research Project: How Available? How Hard? How Dangerous? Proc. of Information Security Curriculum Development Conference, Kennesaw, Georgia, USA, September 28-29, 2007, pp. 43-48
5. CABUK S., DALTON C. I., Towards Automated Provisioning of Secure Virtualized Networks. Conference on Computer and Communications Security. Proceedings of the 14th ACM conference on Computer and communications security. p235 – 245. 2007
6. CHERKAOUI O. HALIMA E. Network virtualization under user control. International Journal of Network Management. Volume 18 , Issue 2 (March 2008) p147 - 158
7. DAVIS, M. A.; BODMER, S.; LEMASTERS, A. Hacking Exposed Malware And Rootkits Malware & Rootkits Secrets & Solutions. 2009. 400p. ISBN 0071591184 / 9780071591188
8. GARFINKEL T., ROSENBLUM M. A Virtual Machine Introspection Based Architecture for Intrusion Detection, ACM SIGOPS Operating Systems Review, Volume 37 , Issue 5 (December 2003).
9. GARFINKEL T., ROSENBLUM M. When Virtual is Harder than Real: Security Challenges in Virtual Machine Based Computing Environments. Proceedings of the 10th conference on Hot Topics in Operating Systems - Volume 10. 2005
10. GARFINKEL T., PFAFF B., CHOW J., ROSENBLUM M., AND BONEH D.. TERRA: A Virtual-Machine Based Platform for Trusted Computing. In Proceedings of the ACM Symposium on Operating Systems Principles (SOSP), Oct. 2003.
11. GOLDBERG, ROBERT P. (February 1973) (PDF). *Architectural Principles for Virtual Computer Systems*. Harvard University. pp. 22–26.

12. HANSEN J. G. Virtual machine mobility with self-migration. Daktaro tezės. Kopenhagos Universitetas. 2009.
13. HOGLUND, G.; BUTLER, J. Rootkits: Subverting the Windows Kernel. 2005. 352p. ISBN-13: 978-0321294319
14. JIANG X., WANG X., XU D. Stealthy Malware Detection Through VMM-Based „Out-of-the-Box" Semantic View Reconstruction. ACM Transactions on Information and System Security (TISSEC). Volume 13 , Issue 2 (February 2010)
15. JONES S., ARPACI-DUSSEAU A., ARPACI-DUSSEAU R. VMM-based Hidden Process Detection and Identification using Lycosid. Proc. of VEE '08, Seattle, Washington, USA, March 5–7, 2008, pp. 91-100
16. LOBO D., WATTERS P., WU X., RBACS: Rootkit Behavioral Analysis and Classification System wkdd, pp.75-80, 2010 Third International Conference on Knowledge Discovery and Data Mining, 2010
17. LYNCH M. D. UNDERSTANDING VirtSec: An executive overview of server virtualization security issues, including best practices for maintaining your security profile. http://www.dmlynch.com
18. MARSHALL D., REYNOLDS W., MCCRORY D. VMware® and Microsoft® Platforms in the Virtual Data Center. Auerbach Publications, Taylor & Francis Group, 2006.
19. MILLER K., PEGAH M. Virtualization, Virtually at the Desktop. User Services Conference, Proceedings of the 35th annual ACM SIGUCCS conference on User services. Orlando, Florida, USA, 2007
20. ORMANDY T. An Empirical Study into the Security Exposure to Hosts of Hostile Virtualized Environments
21. PETRONI N. L., HICKS M., Automated Detection of Persistent Control-Flow Attacks. Conference on Computer and Communications Security. Proceedings of the 14th ACM conference on Computer and communications security. p103 – 115. 2007.
22. POKHAREL M., PARK J. Cloud computing: future solution for e-governance Proc. of the 3rd International Conference on Theory and Practice of Electronic Governance, Bogota, Columbia, November 10-13, 2009, pp. 409-410
23. RUTKOWSKA J. Introducing Stealth Malware Taxonomy. COSEINC Advanced Malware Labs, Version 1.01, November 2006.
24. THIMBLEBY H., ANDERSON S., AND CAIRNS P., A Framework for Modelling Trojans and Computer Virus Infection, Computer Journal, Vol. 41(7), 1998, pp. 444-458, British Computer Society

25. TOLDINAS J., ŠTUIKYS V., ZIBERKAS G., RUDZIKA D. Rootkit aptikimo eksperimentas virtualioje aplinkoje.Elektronika ir elektrotechnika. – Kaunas: Technologija, 2010. – Nr. x(xx). – P. x-xx
26. QUINH N., TAKEFUJI Y. Towards a Tamper-Resistant Kernel Rootkit Detector. Proc. of SAC'07, Seoul, Korea, March 11-15, 2007, pp. 276 – 283
27. Vasisht V., Lee H. SHARK: Architectural Support for Autonomic Protection against Stealth by Rootkit Exploits IEEE, 2008, pp. 106 – 116
28. VIELER, R. Professional Rootkits. 2007. 360 p. ISBN: 978-0-470-10154-4
29. Volatility karkasas: darbinės atminties artefaktų paieškos ir išsaugojimo įrankis: https://www.volatilesystems.com/default/volatility
30. Volatility karkaso įskiepių sąrašas http://www.forensicswiki.org/wiki/List_of_Volatility_Plugins
31. Volatility įskiepių diegimo instrukcija (Windows) http://volatility.googlecode.com/files/install%20plugins.pdf
32. Python galinga, dinaminė, interpretuojama, programavimo kalba. Python galima parsisiųsti iš: http://www.python.org/download/
33. MinGW, MSYS and mingwPORT Projectų svetainė http://www.mingw.org/
34. HBGary. Flaypaper svetainė https://www.hbgary.com/products-services/flypaper (tikrinta 20100515)
35. VMware Workstation svetainė http://www.vmware.com/products/workstation/ (tikrinta 20100515)
36. FU rootkit svetainė http://www.rootkit.com (tikrinta 2010.05.15)

7. Summary

In the context of virtual environment, The Security problems are highly important. The work presents analysis of malware types and it‘s presence in virtualized environments.

Work also presents some results of experiments that have been carried out within the real virtual machine environment through modeling aiming to identify dependencies between the malware type, called Rootkits, detection time and the virtual machine memory size. Rootkits exploit kernel vulnerabilities and gain privileges (popularity) within any system, virtual or not.

The basic result of the work is as follows:

1) the malware detection methodology for the virtual environment when the memory size of a virtual machine is changing;

2) dependences between the virtual machine memory size and Rootkit detection time.

8. Santrumpų ir terminų žodynas

Terminas	**Paaiškinimas**
VM	Virtuali Mašina (angl. Virtual Machine) – programinės įrangos arba aparataūros pagalba suformuota skaičiavimo resursų erdvė, kurioje dirba Operacinė Sistema.
rootkit	Klastinga piktavališka programinė įranga, skirta užmaskuoti savo ir kitos piktavališkos programinės įrangos egzistavimo operacinėje sistemoje pėdsakus.
TCB - trusted computing base	Patikima skaičiavimų bazė – aparatūros, programinės įrangos ir kitų komponetų, nuo kurių priklauso sistemos saugumas rinkinys
Hook	Sisteminės lentelės įrašas

9. Priedai

9.1. Tarptautinės konferencijos „Elektronika 2010“ dalyvio diplomas

Diplomas pažymintis dalyvavimą 14-oje tarptautinėje konferencijoje „Elektronika 2010“

25 pav. ELEKTRONIKA 2010 Diplomas

9.2. Publikacijai paruoštas straipsnis „Rootkit Detection Experiment within a Virtual Environment“

Šiame priede yra pateikiamas mokslinis straipsnis - „Rootkit Detection Experiment within a Virtual Environment“.

Rootkit Detection Experiment within a Virtual Environment

Jevgenijus Toldinas
Computer Department, Kaunas University of Technology,
Studentų 50, LT-51368, Kaunas, Lithuania, email: eugenijus.toldinas@ktu.lt
Vytautas Štuikys, Giedrius Ziberkas
Software Engineering Department, Kaunas University of Technology,
Studentų 50, LT-51368, Kaunas, Lithuania, ph.: +370 37 300399, email: vytautas.stuikys@ktu.lt, ziber@soften.ktu.lt
Darius Rudzika
Computer Department, Kaunas University of Technology,
Studentų 50, LT-51368, Kaunas, Lithuania, email: darius.rudzika@stud.ktu.lt

Introduction

Today, with the further expansion of Information Communication Technology (ICT), we are moving towards a globally virtualized world. We move from single work station to virtual machine and towards cloud computing [1]. Many peoples are working on the go now: at office, school, home, airport, café and other places. On the other hand, virtual environments are operating under conditions that are constantly affected by threats and danger of virus attacks. In such a case, we must take care of information security and protection against the attacks. In general, computer viruses are products of special software called malware. Malware, as described in [2], is classified into four classes: type 0, type I, type II and type III. According to statistics gathered from Microsoft's Malicious Software Removal Tool, a significant fraction of the malware it encounters consists of stealth rootkits [3]. With respect to the mentioned classification rootkits are malware of type I and type II [2].

A rootkit is a small computer program that stealthily invades an operating system (OS) or its kernel and takes control of the computer [3]. Rootkits are receiving more attention now as they are becoming serious security threats to all classes of computing, including embedded devices, desktop users, and server farm machines. As rootkits hide malware activities, i.e. may run as hidden processes, gain accesses as administrator to system resources and exploit kernel vulnerabilities, it is difficult to detect them.

Basically, rootkits can be categorized into two groups: user-level (aka application-level) rootkits and kernel-level rootkits [4, 5]. If user-level rootkits are quite easy to uncover, because they do not modify the OS kernel, the second group poses a lot of problems because the rootkits modify the OS kernel to provide the faked information.

The aim of the paper is to present some framework to investigate the behavior of the kernel-level rootkits and describe the results of experiments we have carried out aiming to model the processes of detection of such kind of malware. The basic result we have identified is the rootkits detection time dependencies upon the virtual machine memory size.

Problem motivation

The typical structure of virtual environment consists of hardware with host operating system (OS), virtualization layer and series (pool) of virtual machines (VM) with guest OS as it is presented in Fig. 1, a).

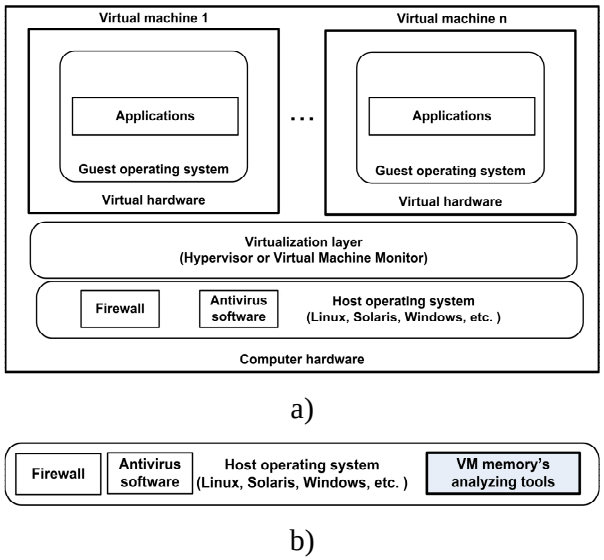


Fig. 1. Typical virtual architecture with firewall and antivirus software (a) and extra tools to support security (b)

The benefits of virtualization are numerous and include, for example, portability, manageability and efficiency in using of computational resources. But the benefits are not for free: we must consider and evaluate security problems that arrive with virtualization. The typical solution of the problems is the use of the firewall and antivirus software installed in the host operating system (see Fig.1, a)). Though the solution is simple enough, however, it cannot ensure entirely the security of virtual machines from the rootkit attacks (for details, see [6]).

To protect virtual machines from that kind of virus, it is possible to install the firewall and antivirus software in each VM. However, such a solution is rather too complex and lacks of flexibility in terms of efficiency and costs. First, there are extra memory losses in each virtual machine. And next, licenses are need for each virtual application, i.e. in each guest OS. As virtual machines are usually created dynamically the number of required licenses should be anticipated for the maximum meaning that not all of them will be in use.

A question that is often asked is: "Can a virtual machine be used to compromise its host server?" Although no known compromises exist today which could be used to attack a host server from within a virtual machine, it is conceivable that it could be accomplished through the virtualization platform's communication mechanisms between host server and virtual machines used by the platform's guest OS enhancement tools [7, see page.98].

What we propose for the problem solution in this paper is the use of VM memory's scanning tools from outside, e.g. from the host OS (see Fig. 1, b). The tools are storied into the host computer and are operating under host OS control. The tools perform scanning of VM memory aiming to identify the existence of the virus.

Problem representation domain

Depending on the level of exploitation, a rootkit can operate in the user space and the kernel space. Kernel mode rootkits are more detrimental than user mode rootkits as they can obtain unrestricted accesses at the root privilege level and thus can freely manipulate any component of the system via the compromised OS.

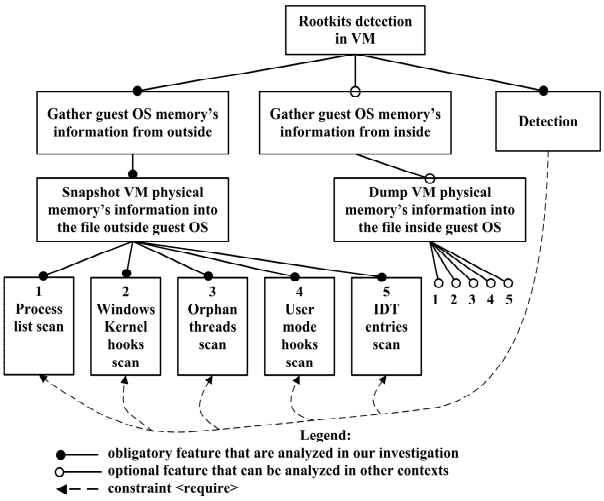


Fig. 2. Problem representation using feature diagram notation

When a virtual machine is running it is possible through the use of existing tools (e.g., memory dump to file) to gather information about memory's (e.g., process list, interrupt description table, kernel mode, user mode, etc.) from both inside and outside of the virtual machine OS.

Fig. 2 explains the features of the problem and solution domains. Note that black circles denote obligatory features that were taken into account in our investigation. White circles denote optional features that can be analyzed in other contexts.

Rootkits detection may be made by the memory scan for known anomalies in the process list, in the kernel mode, in the orphan threads list, in the user mode and in the interrupt descriptor table (IDT).

A virtual machine in the host computer is presented as a continuous file, where the guest OS structure and all information are saved. We provide a brief overview of tools used as follows.

1. **Volatility framework** – The Volatility Framework is a completely open collection of tools, implemented in Python under the GNU (General Public License), for the extraction of digital artifacts from volatile memory (RAM) samples. The extraction techniques are performed completely independent of the system being investigated but offer unprecedented visibility into the runtime state of the system [8].
2. **Volatility plug-ins** – list of the published plug-ins for the Volatility framework [9, 10].
3. **Python** – is an object-oriented, interpreted, and interactive programming language [11].
4. **MinGW** – a port of the GNU Compiler Collection (GCC), and GNU Binutils, for use in the development of native Microsoft Windows applications. In our case the tool was used for developing Volatility plugins [12].
5. **Flypaper.exe** – HBGary Flypaper is loaded as a device driver and it blocks all attempts to exit a process, end a thread, or delete the memory. All components used by the malware remain as a resident in the process list and stay in the physical memory. The entire execution chain is reported so that you can follow each step. HBGary Flypaper is free for non-commercial use [13].
6. **VMware Workstation v6.5 and v7** – Desktop oriented virtualization platform [14].

Strategy of the methodology for the given platform

The proposed methodology contains three stages: initialization, snapshot and analysis. The first stage needs both the guest OS and host OS initial initialization. Each guest OS needs to be initialized with:

1. Rootkit for simulation process (we use FU - open source rootkit with well known code and behavior [15]).
2. Network services (such as, ***tfpd***, ***ftpd***, ***syslogd***) for communication with host OS.

Host OS needs to be initialized with:

1. **Python** runtime v2.6 serves to enable volatility framework run because it is entirely written in that language.
2. **Volatility framework**.
3. **Volatility framework plug-ins**.
4. Microsoft ® **PowerShell**.

The second stage serves for taking guest OS memory's information snapshots periodically and saving that information into the file VMEM (meaning 'virtual memory' for short) outside guest OS for the next stage. Actions of the snapshot stage are detailed in Fig.3.

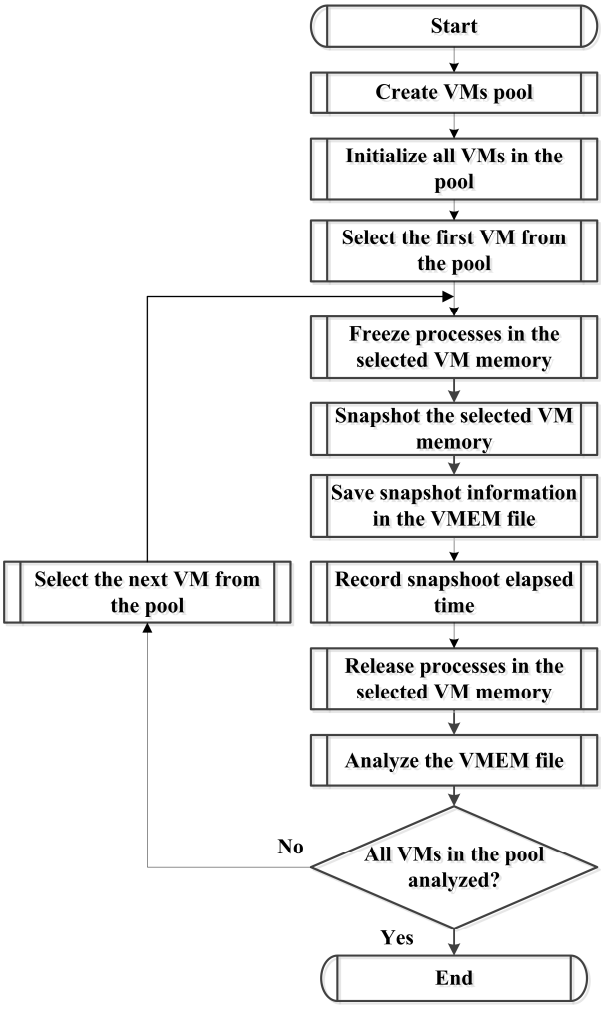


Fig. 3. Algorithm that models ROOTKIT detection functionality within the VM memory's analyzing tools (see Fig.1. b))

It is important to state that we need to freeze the processes within the guest OS for taking the snapshot. At the end of snapshot we record the elapsed time and, than the guest OS processes are released.

The snapshot what was saved into the file identified as VMEM is used at the last stage: analysis for rootkit detection. Fig. 4 presents main processes of the last stage.

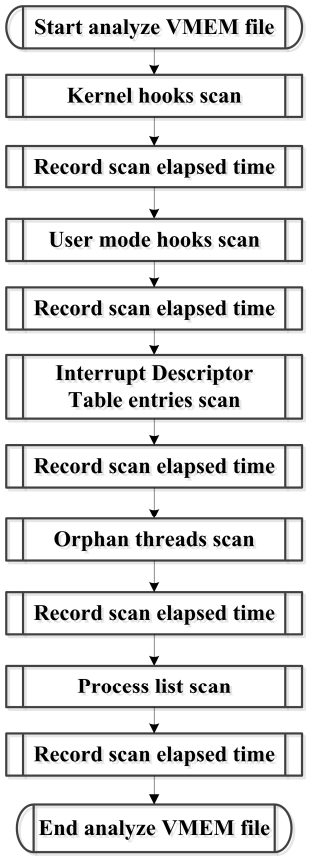


Fig. 4. Analyzing processes for communication and elapsed time recording

We applied five scans for rootkit detection and every scan elapsed time was recorded. The task of mentioned measures was to evaluate and detect how match time takes every scan period and how scan times depends upon guest OS memory's size.

Scripts were written for each scan processes and scan times recording. For all collected snapshots volatility framework was run from **PowerShell**. The number of command lines used for getting experiment results is shown in Table 1.

Table 1. Scripts for snapshot scanning

Analyze purpose	Script
Process list scan	python volatility.py pslist -f '$FULL_PATH_TO_VM_MEMORY_IMAGE.VMEM'
User mode hooks scan	python volatility.py usermode_hooks -f '$FULL_PATH_TO_VM_MEMORY_IMAGE.VMEM' –d C:\temp
Orphan threads scan	python volatility.py orphan_threads -f '$FULL_PATH_TO_VM_MEMORY_IMAGE.VMEM'
Kernel hooks scan	python volatility.py kernel_hooks -f '$FULL_PATH_TO_VM_MEMORY_IMAGE.VMEM' –d C:\temp
IDT entries scan	python volatility.py idt_entries -f '$FULL_PATH_TO_VM_MEMORY_IMAGE.VMEM'

Experiments

The aim of our experiment was to collect experimental data of real VM and to evaluate the proposed methodology by recording the scan time's dependencies upon the VM memory size.

In our experiments, the host machine is Intel P9400 running host **Windows 7** 32-bit OS. The VMM hypervisor is **VMware Workstation v6.5** [14].

The pool of the guest OS was implemented using **Windows XP SP3** 32-bit with the different sizes of random access memory (RAM). The first VM RAM size was 128MB, the second 256 MB, and the next 384MB and so on till the last 2048MB.

In this section we present the results of using our verification function for kernel rootkits detection and give a brief evaluation of verification performance.

The experiment results for each increment of the guest OS memory size, i.e. VM, are presented in Table 2.

Table 2. Experiment result details

VM memory size MB	Snapshot time s	Orphan threads scan s	Process list scan s	Kernel hooks scan s	User mode hooks scan s	IDT entries scan s	Total time s
128	1	3,88	0,14	2,65	127,10	0,77	135,55
256	1	1,12	0,14	2,59	119,72	0,33	124,91
384	1	4,63	0,14	2,57	116,88	0,36	125,59
512	2	6,17	0,14	2,58	116,21	0,42	127,52
640	2	6,70	0,16	2,56	100,38	0,49	112,28
768	2	7,79	0,14	2,58	109,60	0,38	122,50
896	2	8,79	0,14	2,53	108,99	0,50	122,96
1024	19	22,34	0,14	2,56	101,38	0,54	145,97
1152	22	25,93	0,14	2,60	97,35	0,79	148,80
1280	26	27,86	0,14	2,59	108,73	0,56	165,88
1408	26	28,58	0,14	2,53	113,36	0,52	171,14
1536	33	30,72	0,14	2,60	116,85	0,53	183,83
1664	27	33,05	0,14	2,58	121,85	0,56	185,19
1792	33	39,82	0,14	2,55	108,97	0,48	184,96
1920	44	43,66	0,15	2,59	113,93	0,55	204,87
2048	55	46,58	0,14	2,58	117,96	0,40	222,67

In Fig. 5, we present a graphical relationship between the VM memory size and the total time consumed for the snapshot and all types of the VM memory scans.

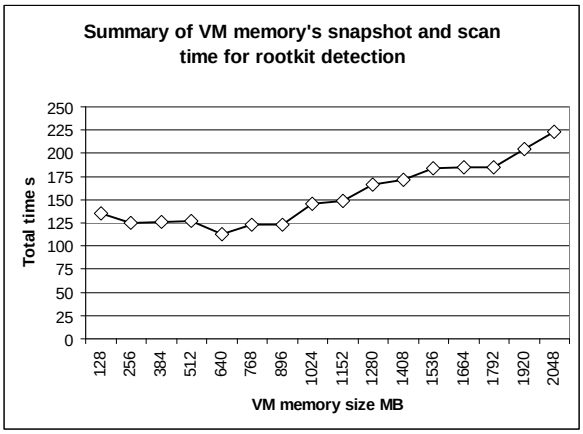
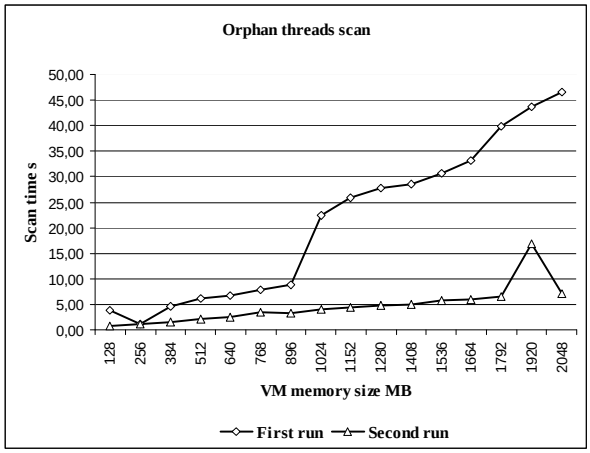
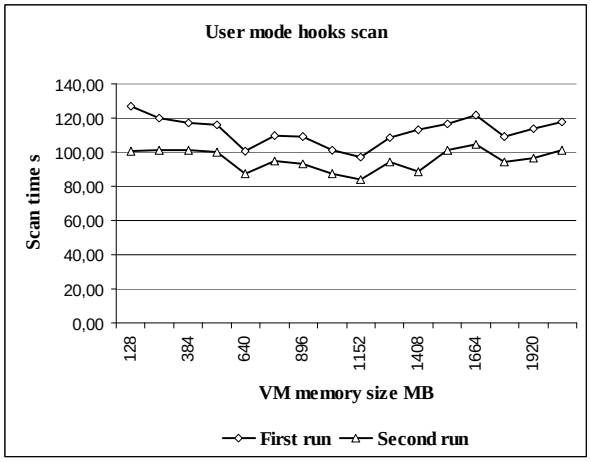


Fig. 5. Total analyze time in VM with various memory sizes

We have also identified that there is a difference in time when we are repeating the scanning process, for example, for the orphan threads scan and for the user mode hooks scan. As it is depicted in Fig. 6, the second scan requires much less time then the first.



a)



b)

Fig. 6. Two runs (a) of the orphan threads scan, (b) of the user mode hooks scan

We explain the phenomena by the memory data cashing effect. Comments on the modelling results given in Table 2 are as follows. Process list, Kernel hooks and IDT entries scans are approximately constant because they are independent upon VM memory size (they depend on guest OS initial configuration only). Small discrepancies are due to the measurement inaccuracy. User mode hooks scan varies depending on the VM workload. In the rest scans times vary depending on the VM memory size, but there is a noticeable jump when the VM memory size reaches 1024 MB. The jump is due to the hypervisor's specificity: when the VM memory size is less than 1024 MB VM are still under operation, and when the size of VM is more or equal to 1024 MB VM are interrupted until the end of the scans.

Conclusions

Our experiment shows that the VM memory size is influential on the total elapsed time for rootkits detection in the following manner: 1) with the increase of memory size the snap shot time grows largely but there is no evident relationship (e.g., if the size of memory changes by factor 2-3 the time increases by factor 15-20); 2) the memory size is most influential on scanning time of Orphan threads, again there is no clearly identifiable

expression of the relationship; 3) Process list and Kernel hooks scan times are practically independent upon the memory size of virtual machines; 4) user mode hooks and IDT scan times not much depend on the memory size, though one can observe some discrepancies in time values. Our experiment results also show that the difference between the minimal amount of time and the maximal amount of time can be expressed by factor 2.

References

[1] **Pokharel M., Park J.** Cloud computing: future solution for e-governance // Proc. of the 3rd International Conference on Theory and Practice of Electronic Governance, Bogota, Columbia, November 10-13, 2009, pp. 409-410

[2] **Rutkowska J.** Introducing Stealth Malware Taxonomy // COSEINC Advanced Malware Labs, Version 1.01, November 2006.

[3] **Bowman M., Brown H., Pitt P.** An Undergraduate Rootkit Research Project: How Available? How Hard? How Dangerous? // Proc. of Information Security Curriculum Development Conference, Kennesaw, Georgia, USA, September 28-29, 2007, pp. 43-48.

[4] **Jones S., Arpaci-Dusseau A., Arpaci-Dusseau R.** VMM-based Hidden Process Detection and Identification using Lycosid // Proc. of VEE '08, Seattle, Washington, USA, March 5–7, 2008, pp. 91-100.

[5] **Quinh N., Takefuji Y.** Towards a Tamper-Resistant Kernel Rootkit Detector // Proc. of SAC'07, Seoul, Korea, March 11-15, 2007, pp. 276 - 283.

[6] **Vasisht V., Lee H.** SHARK: Architectural Support for Autonomic Protection against Stealth by Rootkit Exploits // IEEE, 2008, pp. 106 - 116.

[7] **Marshall D., Reynolds W., McCrory D.** VMware® and Microsoft® Platforms in the Virtual Data Center // Auerbach Publications, Taylor & Francis Group, 2006.

[8] The Volatility Framework: Volatile memory artifact extraction utility framework // https://www.volatilesystems.com/default/volatility

[9] List of Volatility Plugins // http://www.forensicswiki.org/wiki/List_of_Volatility_Plugins

[10] Installing Volatility Plugins (Windows) // http://volatility.googlecode.com/files/install%20plugins.pdf

[11] Python is a remarkably powerful dynamic programming language. Download Python // http://www.python.org/download/

[12] Home of the MinGW, MSYS and mingwPORT Projects // http://www.mingw.org/

[13] HBGary. Flaypaper // https://www.hbgary.com/products-services/flypaper

[14] VMware Workstation // http://www.vmware.com/products/workstation/

[15] **Biancuzzi F.** Windows rootkits come of age // http://www.securityfocus.com/columnists/358/1

Received 2010 03 05

J. Toldinas, V. Štuikys, G. Ziberkas, D. Rudzika. Rootkit detection experiment within a virtual environment // Electronics and Electrical Engineering. – Kaunas: Technologija, 2010. – No. x(xx). – P. x-xx.

In the context of virtual environments, the security problems are highly important. The paper presents some results of experiments we have carried out within the real virtual machine environment through modeling aiming to identify dependencies between the virus, called Rootkits, detection time and the virtual machine memory size. Rootkits exploit kernel vulnerabilities and gain privileges (popularity) within any system, virtual or not. The basic result of the paper is as follows: 1) the Rootkits detection methodology for the virtual environment when the memory size of a virtual machine is changing; 2) dependences between the virtual machine memory size and Rootkit detection time. Ill. 6, bibl. 15, tabl. 2 (in English; abstracts in English, Russian and Lithuanian).

Е. Толдинас, В. Штуйкис, Г. Зиберкас, Д. Рудзика. Эксперимент определения Rootkit в виртуальной среде // Электроника и электротехника. – Каунас: Технология, 2010. - № x(xx). – С. x-xx.

В среде виртуальных машин очень важна проблема безопасности. В статье представлены некоторые результаты эксперимента, используя моделирование, проведенное нами в конкретной среде виртуальных машин, с целью определения зависимости времени обнаружения вирусов типа Rootkits от размера оперативной памяти виртуальной машины. Данные вирусы поражают функциональные свойства ядра операционной системы, широко распространены в различных операционных системах, как в стационарных, так и в виртуальных. Основные результаты: 1) предложена методика обнаружения вирусов типа Rootkits в среде виртуальных машин, при измененяющемся обьеме оперативной памяти; 2) установлена связь между временем обнаружения вируса и размером памяти виртуальных машин. Ил. 6, библ. 15, табл. 2 (на английском языке; рефераты на английском, русском и литовском яз.).

J. Toldinas, V. Štuikys, G. Ziberkas, D. Rudzika. Rootkit aptikimo eksperimentas virtualioje aplinkoje // Elektronika ir elektrotechnika. – Kaunas: Technologija, 2010. – Nr. x(xx). – P. x-xx.

Saugumo problema virtualių mašinų aplinkoje yra labai svarbi. Šiame straipsnyje pateikiami kai kurie eksperimentiniai rezultatai, kuriuos mes gavome konkrečioje virtualių mašinų aplinkoje modeliavimo būdu siekdami nustatyti virusų, žinomų Rootkits vardu, aptikimo laiką priklausomai nuo virtualios mašinos atminties dydžio. Šie virusas klastingai pažeidžia operacinės sistemos branduolio funkcionalumą, yra smarkiai paplitę bet kurioje sistemoje, virtualioje arba stacionarioje. Pagrindinis straipsnio rezultatas toks: 1) pasiūlyta virusų Rootkits aptikimo metodika virtualių mašinų aplinkoje, kai keičiasi virtualių mašinų operatyviosios atminties dydis; 2) nustatytos viruso aptikimo laiko priklausomybės nuo virtualios mašinos atminties dydžio. Il. 6, bibl. 15, lent. 2 (anglų kalba; santraukos anglų, rusų ir lietuvių k.).