TECHNICKÁ UNIVERZITA V KOŠICIACH

FAKULTA ELEKTROTECHNIKY A INFORMATIKY

RIEŠENIE BEZPEČNOSTNEJ POLITIKY V PROSTREDÍ MS WINDOWS

Diplomová práca

2014 **Bc. Martin Hančák**

TECHNICKÁ UNIVERZITA V KOŠICIACH

FAKULTA ELEKTROTECHNIKY A INFORMATIKY

RIEŠENIE BEZPEČNOSTNEJ POLITIKY V PROSTREDÍ MS WINDOWS

Diplomová práca

Študijný program:	Informatika
Študijný odbor:	9.2.1 Informatika
Školiace pracovisko:	Katedra počítačov a informatiky (KPI)
Školiteľ:	Ing. Anton Baláž, PhD.
Konzultant:	Ing. Anton Baláž, PhD.

2014 Košice **Bc. Martin Hančák**

Abstrakt v SJ

Cieľom diplomovej práce je vytvorenie systémového pieskoviska na báze modelu virtualizácie na úrovni operačného systému v prostredí Microsoft Windows. Systémové pieskovisko umožňuje vytvoriť bezpečné prostredie pre nedôveryhodné aplikácie. Práca obsahuje základné informácie o problematike systémových pieskovísk a ich vzťahu k virtualizácii. Pozornosť je tiež venovaná analýze súčasnému stavu problematiky vrátane rozboru existujúcich riešení. Navrhnuté pieskovisko kontroluje prístup k súborom a adresárom, registrom Windows, obmedzuje systémové zdroje a prístup na internet. Operácie na súboroch filtruje ovládač v režime jadra implementovaný ako minifilter. Celkový prístup je riadený na báze kopírovania súboru otvoreného s príznakom zápisu. Registrové operácie sú prerušované v ovládači pomocou rozhrania pre filtrovanie registrov. Logika rozhodovania je pri registroch riadená modelom kopírovania kľúčov a hodnôt pri vytvorení kľúčov resp. zmene hodnoty. Obmedzenie systémových zdrojov a prístupu na sieť spravuje služba operačného systému spustená v režime používateľa. Systémové pieskovisko je ovládane pomocou samostatnej aplikácie s jednoduchým grafickým rozhraním. Výsledkom práce je funkčný softvér pre zabezpečenie operačného systému Windows pred potenciálne nebezpečnými aplikáciami spustenými v systémovom pieskovisku. Prínosom práce je výsledné riešenie, ktoré funguje naprieč modernými verziami OS Windows. Pieskovisko minimálne ovplyvňuje sémantiku a čas vykonávaného programu a poskytuje rozhranie pre jednoduché používanie.

Kľúčové slová

systémové pieskovisko, bezpečnosť operačného systému, minifilter, filter registrov, virtualizácia

Abstrakt v AJ

The main goal of this thesis is sandbox creation in the Microsoft Windows environment on OS-level virtualization model. The sandbox creates safe environment for untrusted applications. This thesis contains elementary information about sandboxes and their relations with virtualization. The analysis of current situation of this topic, including existing solutions is also a part of this thesis. Designed sandbox controls file, directory and Windows registry access. It limits system resources and internet access. A kernel level driver, implemented as minifilter, filters file operations. Global access is managed by copy on open for write method. Interface for registry filtering interface interrupts registry operations in this driver. The core of registry filtering is based on copy on write method. User level Windows service restricts system resources and network access. Standalone application with simple graphical user interface controls the sandbox. The result of this thesis is software capable of securing Windows operating system from potentially dangerous applications executed in the sandbox. This software is compatible with all current Windows versions. The semantics and time of program execution are barely affected by sandbox. It offers simple user interface as well.

Kľúčové slová v AJ

sandbox, operating system security, minifilter, registry filter, virtualization

Zadanie práce

TECHNICKÁ UNIVERZITA V KOŠICIACH
FAKULTA ELEKTROTECHNIKY A INFORMATIKY
Katedra počítačov a informatiky

ZADANIE DIPLOMOVEJ PRÁCE

Študijný odbor: **9.2.1 Informatika**
Študijný program: **Informatika**

Názov práce:

Riešenie bezpečnostnej politiky v prostredí MS Windows
MS Windows Security Sandbox

Študent:	**Bc. Martin Hančák**
Školiteľ:	**Ing. Anton Baláž, PhD.**
Školiace pracovisko:	**Katedra počítačov a informatiky**
Konzultant práce:	
Pracovisko konzultanta:	

Pokyny na vypracovanie diplomovej práce:

1.Oboznámiť sa s problematikou systémových pieskovísk a zabezpečenia.
2.Analyzovať súčasný stav problematiky.
3.Praktickou aplikáciou overiť navrhnuté zabezpečenie MS Windows pomocou pieskoviska systémových volaní.
4.Vypracovať dokumentáciu podľa štandardov.

Jazyk, v ktorom sa práca vypracuje: slovenský
Termín pre odovzdanie práce: 02.05.2014
Dátum zadania diplomovej práce: 31.10.2013

.....
doc. Ing. Jaroslav Porubän, PhD.
vedúci garantujúceho pracoviska

TECHNICKÁ UNIVERZITA V KOŠICIACH FAKULTA ELEKTROTECHNIKY A INFORMATIKY -11-

.....
prof. Ing. Liberios Vokorokos, PhD.
dekan fakulty

Čestné vyhlásenie

Vyhlasujem, že som celú diplomovú prácu vypracoval/a samostatne s použitím uvedenej odbornej literatúry.

Košice, 28. apríl 2014 .....

vlastnoručný podpis

Poďakovanie

Na tomto mieste by som sa chcel poďakovať vedúcemu mojej diplomovej práce Ing. Antonovi Balážovi, PhD. za hodnotné rady, odbornú pomoc a usmernenie pri vypracovaní práce. Zároveň sa chcem poďakovať svojej rodine za plnú podporu počas môjho štúdia na vysokej škole.

Predhovor

Bezpečnosť v operačnom systéme Microsoft Windows je častou témou rôznych prác a špecialistov. Konkrétna špecializácia na doménu systémových pieskovísk je predovšetkým vo svete na vzostupe. Dôvodom je nepochybne rozšírenie internetových technológií, s ktorými súvisí zvyšovanie komplexnosti vytvorených vírusov a iného škodlivého softvéru. Čoraz častejšie využitie pieskovísk tiež môže mať súvis s dostatočným výkonom počítačov, ktoré tak viditeľne nespomaľujú ich bezpečné vykonávanie.

Zámerom mojej práce je vytvorenie softvéru pre zabezpečenie operačného systému Windows. Mojou ambíciou je vytvorenie komplexného riešenia, ktoré by bolo možné jednoducho používať, aplikácie v ňom vykonávane by nemohli infikovať operačný systém a zároveň by nebola ovplyvnená ich činnosť počas vykonávania.

Obsah

Zoznam obrázkov

Zoznam tabuliek

Zoznam symbolov a skratiek

ACE (z angl. **A**ccess **C**ontrol **E**ntries) položky pre riadenie prístupu, elementy v zozname pre riadenie prístupu (Access Control List), monitorujúce prístup k špecifikovanému objektu

API (z angl. **A**pplication **P**rogramming **I**nterface) rozhranie pre programovanie aplikácií resp. zbierka funkcií, tried a procedúr ľubovoľnej knižnice alebo programu, ktoré sú dostupné pre programátora

BIOS (z angl. **B**asic **I**nput **O**utput **S**ystem) základný softvér počítača, vrstva medzi hardvérom a operačným systémom

DLL (z angl. **D**ynamic **L**ink **L**ibrary) koncept spoločne využívaných dynamických knižníc v operačných systémoch Windows

ECP (z angl. **E**xtra **C**reate **P**arameter) štruktúra na ukladanie ľubovoľných údajov, ktorú je možné asociovať s operáciou vytvorenia súboru na disku alebo s operáciou vytvorenia súboru zariadenia

IRP (z angl. **I**/O **R**equest **P**acket) štruktúra reprezentujúca vstupnú/výstupnú operáciu v operačných systémoch Windows

KPP (z angl. **K**ernel **P**atch **P**rotection) alebo PatchGuard, je ochrana pred modifikáciami jadra systému Windows implementovaná v 64 bitových verziách

KMDF (z angl. **K**ernel-**M**ode **D**river **F**ramework) softvérový rámec pre tvorbu ovládačov v režime jadra operačného systému Windows, založený na modeli WDM s dôrazom na jednoduchšiu a robustnejšiu implementáciu

NDIS (z angl. **N**etwork **D**river **I**nterface **S**pecification) programové rozhranie abstrahujúce sieťový hardvér od sieťových ovládačov

PID (z angl. **P**rocess **Id**entifier) číslo jednoznačne identifikujúce proces v operačnom systéme

UAC (z angl. **U**ser **A**ccount **C**ontrol) v slovenskom preklade riadenie používateľských účtov, je technológia prítomná v operačnom systéme Windows od verzie Vista, zvyšujúca bezpečnosť obmedzením oprávnení aplikácií na úroveň používateľa, pokiaľ administrátor nepotvrdí zvýšenie práv aplikácie

UMDF (z angl. **U**ser-**M**ode **D**river **F**ramework) softvérový rámec pre tvorbu ovládačov, alternatíva ku KMDF v rámci režimu používateľa

VHD (z angl. **V**irtual **H**ard **D**isk) špecifikácia formátu virtuálneho disku od firmy Microsoft

VMDK (z angl. **V**irtual **M**achine **D**is**k**) formát súboru vytvorený firmou VMWare pre ich virtualizačné riešenia, v súčasnosti s dostupnou otvorenou špecifikáciou

WCF (z angl. **W**indows **C**ommunication **F**oundation) programové rozhranie v balíku .NET pre vytváranie aplikácií orientovaných na služby

WDF (z angl. **W**indows **D**river **F**ramework) množina knižníc pre tvorbu ovládačov pozostávajúca z dvoch samostatných softvérových rámcov KMDF a UMDF

WDM (z angl. **W**indows **D**river **M**odel) základný model pre tvorbu ovládačov v systémoch Windows

WPF (z angl. **W**indows **P**resentation **F**oundation) grafický podsystém a programové rozhranie v balíku .NET pre tvorbu prezentačnej časti softvéru

XAML (z angl. E**X**tensible **A**pplication **M**arkup **L**anguage) deklaratívny jazyk založený na XML pre použitie primárne v technológii WPF

x86 označenie architektúry založenej na Intel 8086 alebo označenie architektúry kompatibilnej s IBM PC

Slovník termínov

AVL strom je údajová štruktúra pre uchovávanie a vyhľadávanie údajov resp. je to samovyvažovací binárny vyhľadávací strom pomenovaný podľa G. M. Adelson-Velskym a E. M. Landisovi.

EResource je synchronizačný mechanizmus umožňujúci súbežné čítanie a exkluzívny zápis k spoločnému zdroju.

Hašovacia funkcia je funkcia pre konverziu vstupného reťazca na výstupný reťazec, ktorý je tvorený obvykle kratšou číselnou charakteristikou pevnej dĺžky.

Hypervizor je softvér alebo hardvér, ktorý vytvára a spúšťa virtuálne stroje.

Normalizovaný názov alebo dlhý názov je štandardný formát názvu súborov od verzie operačného systému Windows 95. Alternatívou je tzv. krátky názov používaný v operačných systémoch MS-DOS a pre účely kompatibility podporovaný aj v nových verziách operačného systému Windows.

Plug and Play je technológia umožňujúca jednoduché, automatické rozpoznávanie a inštaláciu hardvéru.

Pomenované rúry je jednosmerný alebo duplexný spôsob komunikácie medzi ľubovoľnými procesmi v operačnom systéme.

Prefetch je technológia prvýkrát zavedená v operačnom systéme Windows XP pre zrýchlenie procesu štartovania operačného systému a štartu aplikácií. Rozšírenie od verzie Windows Vista sa označuje pojmom SuperFetch.

Rýchly mutex je synchronizačný mechanizmus pre vzájomne vylučujúci prístup k jednému, spoločnému zdroju.

Úvod

Počítačová bezpečnosť je v súčasnosti aktuálnou témou. Postupom času sa stáva problémom, ktorý potrebuje riešiť stále väčší počet používateľov. Zatiaľ čo v minulosti bola táto otázka prevažne doménou odborníkov, dnes s globálnou dostupnosťou internetu sa počítačovou bezpečnosťou musí zaoberať aj koncový používateľ. Aj keď bezpečnosť založená na princípoch tvorí elementárny základ bezpečného softvéru, nikdy nie je možné garantovať absolútnu bezpečnosť. Ak už útočník prenikne do systému, prípadne pod útočníkom rozumieme chybný softvér spôsobujúci poškodenie dát, existuje snaha minimalizovať škody ním napáchané. Možným riešením pri bezpečnostnom aspekte je použitie antivírusových programov, ktoré využívajú antivírusový test a rôzne heuristické metódy v reálnom čase pre nájdenie infikovaného kódu. Efektivita infikovania operačného systému škodlivým softvérom sa za niekoľko posledných rokov dramaticky zvýšila. Množstvo a komplexnosť trójskych koňov, vírusov alebo spyware každým dňom rastie a účinná ochrana v podobe antivírusových riešení nie vždy zabráni útoku. Tieto konvenčné bezpečnostné balíky spoliehajú v prvom rade na aktualizovanú databázu škodlivého kódu, analyzujú správanie programov počas dynamickej analýzy alebo sa opierajú o statickú analýzu kódu. Uvedené metódy nie sú dokonalé, a preto občas neidentifikovaná hrozba prenikne do operačného systému a infikuje dôležité komponenty.

Odlišnou myšlienkou, a teda uprednostnením prevencie pred reakciou je uplatnenie metódy systémových pieskovísk. Vo všeobecnosti existuje viacero kategórií pieskovísk, z ktorých je každé vhodné pre iné použitie. Prvý typ kompletne izoluje aplikácie so samostatným operačným systémom do nezávislého virtuálneho stroja. Limitovanie množiny zdrojov je typické pre tzv. väzenia resp. bezpečnostné kontajnery, ktoré obsahujú obmedzenia diskových kvót, sieťového prístupu a najmä limitovanie pohybu v strome súborového systému. Ďalšia kategória, vykonávanie založené na vopred definovaných pravidlách, umožňuje používateľom definovať bezpečnostnú politiku pre aplikáciu a ovládať tak zabezpečenie systému. Posledným rozšíreným typom sú pieskoviská na natívnom hostiteľovi, slúžiace predovšetkým na analýzu chovania škodlivého softvéru. S výnimkou poslednej kategórie má väčšina systémových pieskovísk príliš reštrikčnú bezpečnostnú politiku, čo môže spôsobovať chyby počas vykonávania. Na druhej strane, koncept nezávislých virtuálnych strojov má vysoké

nároky na zdroje počítačového systému a softvérové vybavenie. Definovanie pravidiel pre vykonávanie je zase neľahký proces aj pre systémových administrátorov a takmer nemožné pre obyčajných používateľov. Analyzátory chovania škodlivého softvéru sú príliš špecifické nástroje s náročnou konfiguráciou. Problémom väčšiny systémových pieskovísk v operačnom systéme Windows je problémová podpora nových verzií a 64 bitových edícií. Taktiež okrem špecifických nástrojov pre analýzu, pieskoviská v systémoch Windows ťažkopádne alebo vôbec neinformujú o činnosti vykonávaných aplikácií.

Práca oboznamuje čitateľa s problematikou systémových pieskovísk a ich vzťahu k všeobecnému pojmu virtualizácie s výberom najvhodnejšej realizácie. Ďalej analyzuje a hodnotí aktuálne dostupné produkty pre túto problematiku v operačnom systéme Windows. Motiváciou diplomovej práce je návrh systémového pieskoviska s vyriešením spomenutých nedostatkov a ukážka metód pre jeho úspešnú implementáciu. Navrhnuté pieskovisko bude minimálne ovplyvňovať vykonávanie programov z hľadiska ich sémantiky. V praxi to znamená všeobecné povolenie čítania súborov a konfigurácie operačného systému. Modifikácia operačného systému mimo povolené umiestnenie je striktne zakázaná, avšak s vytvorením ilúzie o opaku pre aplikácie v pieskovisku, následkom čoho je dosiahnutá nižšia úroveň reštrikcie. Vďaka štandardným rozhraniam pre úpravu systémových volaní dostupných v operačnom systéme Windows, riešenie musí byť bezproblémovo kompatibilné s najnovšími verziami OS a do budúcnosti pripravené pre potrebné modifikácie. Cieľom je vytvoriť komplexný bezpečnostný softvér pre zabezpečenie operačného systému Windows vrátané súborov, registrov, systémových zdrojov a sieťového prístupu. Systémové pieskovisko má predísť infikovaniu operačného systému škodlivým softvérom a taktiež zabrániť poškodeniu OS, ktoré spôsobí napr. chybný program. Motiváciou tiež bola snaha o vytvorenie jednoduchého a nenáročného ovládania pieskoviska bez nutnej konfigurácie a možnosťou práce s aplikáciou cez grafické rozhranie.

V súčasnosti je na osobných počítačoch dominantný operačný systém Windows od firmy Microsoft. Alternatívne operačné systémy okrem menej častých útokov, ktorých dôvodom je menší podiel na trhu, väčšinou štandardne obsahujú rôzne variácie systémových pieskovísk pre limitovanie množiny zdrojov a definovanie pravidiel bezpečnostnej politiky. Návrh systémového pieskoviska v diplomovej práci úzko súvisí

s operačným systémom Windows, avšak niektoré použité metódy sa dajú využiť aj u alternatívnych operačných systémoch.

1 Analýza problematiky systémových pieskovísk a ich vzťah k virtualizácii

Systémové pieskovisko zahrňuje široké spektrum programov. Konkrétna definícia závisí od typu riešenia. Vcelku univerzálne ho vymedzil Hoopes [1], ktorý pod pojmom *sandbox*, v doslovnom preklade pieskovisko, popísal softvér poskytujúci monitorované a kontrolované prostredie, v ktorom ľubovoľný neznámy program nemôže svojím vykonávaním spôsobiť žiadne škody systému, na ktorom reálne beží.

Pieskovisko v užšom zmysle slova umožňuje spúšťať aplikácie takým spôsobom, pri ktorom nie je povolené týmto aplikáciám čítať a zapisovať údaje mimo prípustnej cesty t.j. mimo pieskoviska. V širšom zmysle slova je potrebné k tejto definícii pripojiť kontrolu a prideľovanie zdrojov operačného systému vrátane sieťových služieb, správy hardvéru, nízko-úrovňového prístupu a podobne.

Koncept kontrolovaného vykonávania prvýkrát zaviedol R. Wahbe et al. [2] v kontexte ochrany proti softvérovým chybám. *Sandboxing*, ako označili túto techniku, je spôsob izolácie nedôveryhodných modulov vykonávaných v rovnakom adresnom priestore ako dôveryhodné moduly s minimálnym dopadom na čas vykonávania.

1.1 Virtualizácia

V úvode bolo načrtnuté základné členenie systémových pieskovísk. Takmer každý typ pieskoviska sa nejakým spôsobom vzťahuje na technológiu virtualizácie. Okrajovo môžu pôsobiť riešenia založené na modeli povinného riadenia prístupu napr. SELinux v operačnom systéme Linux. Vo všeobecnosti bezpečnosť, zabezpečenie systému a problematika systémových pieskovísk úzko súvisí s virtualizáciou. Preto je dobré na začiatok vysvetliť, čo v skutočnosti virtualizácia je a aké sú existujúce spôsoby virtualizácie. Pod pojmom virtualizácia sa označujú postupy a techniky, ktoré umožňujú rozdeliť dostupné zdroje počítača do niekoľkých prostredí. Virtualizované prostredie môže byť jednoduchšie prispôsobené potrebám používateľov, ľahšie sa používať, prípadne pred používateľmi zakrývať pre nich nepodstatné detaily, napr. fyzické rozmiestnenie hardvérových prostriedkov.

S virtualizáciou sa vzťahuje aj virtuálny stroj (VM). Virtuálny stroj predstavuje nezávislé prostredie resp. softvérovú implementáciu stroja, ktorý vykonáva programy podobne ako fyzický počítač [3].

Virtualizovať je možné na rôznych úrovniach, od celého počítača až po jeho jednotlivé hardvérové komponenty, prípadne iba vymedzené softvérové prostredie. V rámci tohto prístupu je podľa Barretta a Kippera [4] možné rozdeliť virtualizáciu na tieto hlavné kategórie:

- emulácia,
- plná virtualizácia,
- natívna virtualizácia,
- paravirtualizácia,
- virtualizácia na úrovni operačného systému.
- aplikačná virtualizácia.

Týchto šesť základných techník predstavuje hlavné kategórie a najčastejšie spôsoby virtualizácie. Okrem nich existujú aj ďalšie typy virtualizácie:

- sieťová virtualizácia,
- virtualizácia dátových úložísk.

V nasledujúcich častiach sú stručne popísané hlavné metódy virtualizácie v nadväznosti na tematiku systémových pieskovísk, ich princíp, výhody a implementačné riešenia.

1.1.1 Emulácia

Virtuálny stroj kompletne emuluje skutočný hardvér, prípadne hardvér taký, ktorý v reálnom svete neexistuje. Emulácia predstavuje najstarší spôsob virtualizácie a je niekedy označovaná ako „trap and emulate“ alebo binárna virtualizácia [5]. Virtuálny stroj emuluje hardvér a umožňuje v tomto prostredí používať nemodifikovaný operačný systém.

Pri emulácii je vykonávanie inštrukcií rádovo pomalšie ako pri natívnom vykonávaní. Emulácia sa uplatňuje pri návrhu nového hardvéru, vytváraní softvéru pre hardvér, ktorý nie je fyzicky dostupný, pri potrebe spúšťania aplikácií, pre ktoré už hardvér napr. z dôvodu zastarania nie je možné získať a podobne. Model emulácie vo všeobecnosti nie je vhodný pre problematiku systémových pieskovísk a zabezpečenia. Príkladom emulátorov sú Bochs, Qemu alebo Microsoft Virtual PC pre architektúru PowerPC.

1.1.2 Plná virtualizácia

Plná virtualizácia je technika používaná v prostredí virtuálneho stroja, ktorý simuluje základný hardvér. Platí tu pravidlo, pri ktorom každý operačný systém podporovaný na základnom hardvéri môže byť spúšťaný v každom VM. Používatelia môžu simultánne spúšťať prakticky ľubovoľný počet hosťujúcich operačných systémov. Pri plnej virtualizácii virtuálny stroj dostatočne simuluje hardvér na to, aby bolo možné hosťujúce OS izolovane spúšťať bez dodatočných modifikácií. Virtualizovaný OS si nie je vedomý skutočnosti, že sa vykonáva vo VM. Takýto prístup predstavuje množstvo výhod, napr. pri vývoji operačných systémov, kedy je izolácia a bezpečnosť kľúčový faktor [6].

Hypervizor resp. monitor virtuálnych strojov, poskytuje pre každý VM všetky potrebné služby vrátane virtuálneho BIOS-u, virtuálnych zariadení a správy virtuálnej pamäte. Virtualizovaný operačný systém nemá priamy prístup k základnej vrstve hardvéru iba k tzv. virtualizačnej vrstve. Plnú virtualizáciu napríklad umožňuje Microsoft Virtual PC.

1.1.3 Natívna virtualizácia

Najnovšia technológia virtualizácie v rámci architektúry x86, často označovaná ako hybridná alebo hardvérovo-asistovaná virtualizácia. Je kombináciou plnej virtualizácie a paravirtualizácie. Hypervizor ma jednoduchšiu a robustnejšiu implementáciu [7]. Využíva najnovšie inštrukčné sady posledných generácií CPU (od roku 2005), konkrétne technológie Intel VT-x a AMD-V. V súčasnosti je to dominantný spôsob virtualizácie v oblasti osobných počítačov a čiastočne aj u serverových riešení. Implementáciu natívnej virtualizácie podporujú nástroje VMware Workstation, Xen 3.x, Oracle VirtualBox a Microsoft Hyper-V.

1.1.4 Paravirtualizácia

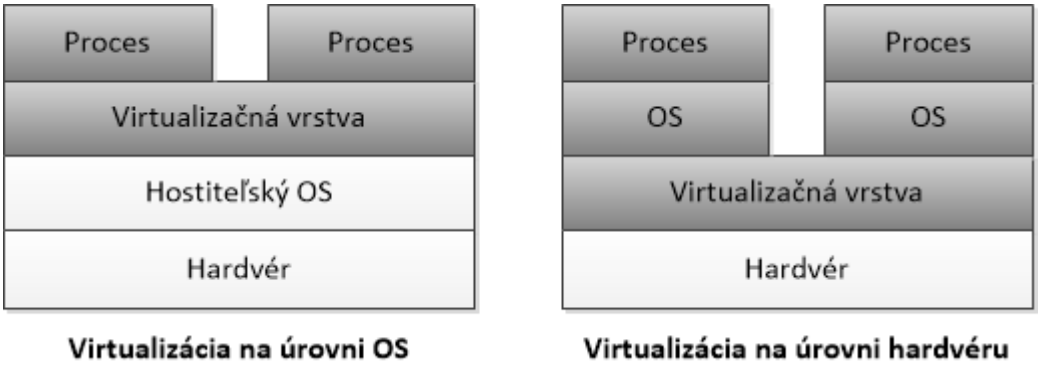
Jedna z najnovších technológií v oblasti virtualizácie. Výhodou je lepší výkon na rozdiel od plnej virtualizácie, ale za cenu modifikácie jadra OS klienta. Podobne ako pri plnej virtualizácii, paravirtualizácia využíva hypervizor. Hypervizor je softvér, ktorý rozhoduje o pridelení pamäti, CPU, disku a sieťových zdrojov pre jednotlivé virtuálne stroje. Hypervizor taktiež poskytuje rozhranie pre ostatné systémové operácie vrátane správy pamäte a obsluhy prerušení.

Modifikácia hostiteľského OS umožňuje koordinovanú komunikáciu medzi hypervizorom a virtuálnym strojom, čo vedie k redukcii použitých privilegovaných inštrukcií zodpovedných za pokles výkonu u plnej virtualizácii. [8]

Paravirtualizácia môže hlavne v podnikovom prostredí vyvolať určité otázky, pretože vyžaduje zásahy do jadra OS. V porovnaní s nasadením plnej virtualizácie sú jednoduchšie. Nevýhodou je horšia implementácia u systémov bez dostupného zdrojového kódu. Projekt XEN je príkladom paravirtualizácie v systémoch Linux.

1.1.5 Virtualizácia na úrovni operačného systému

Uvedené predchádzajúce spôsoby predstavovali hardvérovú virtualizáciu, kedy je simulovaný kompletný hardvér alebo niektoré časti hardvéru pre potreby spúšťania nemodifikovaného alebo modifikovaného klientskeho OS. Porovnanie virtualizácie na hardvérovej úrovni a virtualizácie na úrovni OS ilustruje Obr. 1.



Obr. 1 Porovnanie virtualizácie na úrovni OS a hardvéru [9]

Virtualizácia na úrovni OS (ďalej len OS virtualizácia) rozdeľuje zdroje fyzického stroja na úrovni operačného systému [10]. Virtualizované aplikácie sa priamo nevykonávajú v hosťovskom OS, ale využívajú služby jadra hostiteľského OS prostredníctvom virtualizačnej vrstvy. Hostiteľský OS resp. jeho jadro dovoľuje používať viacnásobné izolované hosťovské OS, prípadne iba izolované prostredia pre beh programov. Tieto inštancie sa nazývajú softvérové väzenia, kontajnery, virtuálne prostredia (VE) alebo virtuálne privátne servery (VPS) [9].

Medzi výhody virtualizácie na úrovni OS patrí nízka réžia, čo umožňuje efektívny beh viacerých VM, transparentná migrácia aplikácií, jedna inštalácia OS pre jednoduchú správu a aktualizácie, takmer natívna rýchlosť, podpora hardvéru a funkcií hostiteľského OS a vylepšená bezpečnosť systému. Nevýhody zahrňujú skutočnosť, že OS virtualizácia nie je taká flexibilná ako virtualizácia na úrovni hardvéru [11]. Okrem

iného to znamená, že nie je možné súčasne virtualizovať rôzne typy OS ako sú Windows a Linux.

Príklady programov pre virtualizáciu na úrovni OS sú Linux-VServer, Parallels Virtuozzo Containers, FWM, FreeBSD Jails, Sandboxie a Solaris Containers.

1.1.6 Aplikačná virtualizácia

Aplikačná virtualizácia chápe virtuálne stroje ako menšie nezávislé aplikácie, ktoré umožňujú emulovať špecifické prostredie na systéme klienta [12]. Jednoduchšie povedané, izoluje aplikácie od operačného systému. Pri aplikačnej virtualizácii je aplikácia najčastejšie dodávaná vo forme jedného spustiteľného súboru. Existuje viacero foriem aplikačnej virtualizácie, z ktorých najbežnejšie sú „aplikácie pieskoviska“ a „aplikačný streaming“ [13].

Aplikácie pieskoviska riešia napr. v systémoch Windows problémy s rôznymi verziami DLL, registrovým prístupom a podobne. Nevyžadujú inštaláciu, nezasahujú do hostiteľského OS a vzájomne sa neovplyvňujú [14]. To znamená dobrú prenositeľnosť, možnosť využívať staršie aplikácie v novších verziách OS resp. simultánne používať viacero verzií rovnakého softvéru. Aplikačnú virtualizáciu vo forme aplikácie pieskoviska napríklad podporujú programy Cameyo alebo Evalaze.

Aplikačný streaming zahrňuje lokálne zariadenie a aplikáciu nainštalovanú vo vzdialenom stroji. Lokálny stroj poskytuje fyzické zdroje potrebné pre beh aplikácií, ale tieto aplikácie počítač neobsahuje nainštalované. Aplikácia je „vysielaná“ zo servera po častiach, ktoré sú aktuálne potrebné. Po ukončení práce s programom, sú všetky dočasné komponenty odstránené. Príkladom implementácie môžu byť riešenia Microsoft Application Virtualization a Citrix XenApp.

1.2 Pieskovisko a virtualizácia

Rozdielne typy virtualizácie sa môžu líšiť v sile izolácie, požiadavkách na zdroje, výkonnostnej réžii, škálovateľnosti a flexibilite. Vo všeobecnosti, čím je virtualizačná vrstva „bližšie“ k hardvéru, vytvorené virtuálne stroje sú navzájom lepšie izolované a oddelené od hostiteľského PC, ale na druhej strane potrebujú viac zdrojov a sú menej flexibilné. Medzitým čo je virtualizačná vrstva „ďalej“ od hardvéru, rastie výkon a škálovateľnosť virtuálnych strojov.

Ako už možno vyplynulo z popisu jednotlivých kategórií, virtualizácia pri správnom použití prináša množstvo výhod, ktoré by sa podľa Hoopesa [1] dali popísať

troma bodmi: *konsolidácia*, *spoľahlivosť* a *bezpečnosť*. Konsolidácia zahrňuje efektívnejšie využitie počítačov, zjednodušuje migráciu na novšie verzie rôznych systémov, výrazne skracuje vývoj a testovanie a umožňuje na jednej fyzickej platforme hostiť rozličné OS. Spoľahlivosť sa viac ako kedykoľvek predtým stala jednou z priorít pre mnoho IT spoločností. Má priamy vzťah s dostupnosťou systému, dobou prevádzky a tým pádom aj príjmami firmy. Systémová chyba vo virtuálnom stroji nemá vplyv na ostatné časti systému na rovnakej hardvérovej platforme, čím je zaručená spoľahlivosť a konzistentnosť systému ako celku. Rovnaká technológia, ktorá poskytuje ochranu pred chybami aplikácií tiež poskytuje izoláciu proti bezpečnostným chybám. Pokiaľ by bola narušená bezpečnosť konkrétnej časti virtuálneho stroja, vďaka izolácii je možné kompromitovanú jednotku bez problémov odstaviť. Okrem iného býva dovolené pre každý oddiel aplikovať rozličnú bezpečnostnú politiku, takže pre kritické časti systému vystavené možným bezpečnostným útokom je umožnené v závislosti od potreby prispôsobovať stupeň ochrany.

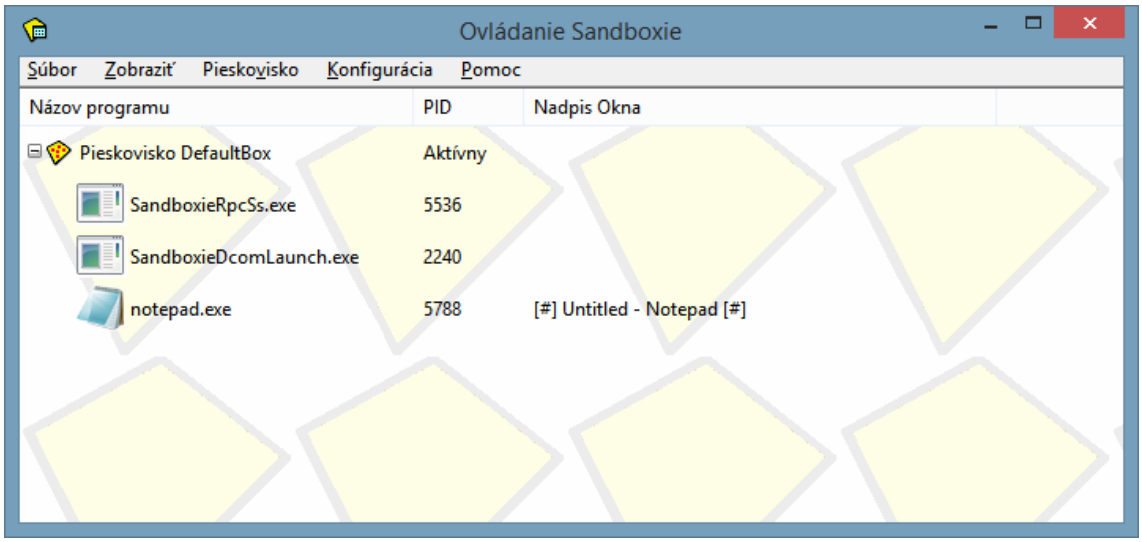
Ako súvisí pieskovisko s virtualizáciou? Podľa charakteristík jednotlivých spôsobov virtualizácie a definície uvedenej v časti 1.1, *systémové pieskovisko predstavuje špecifický prípad virtuálneho stroja.* Podobne ako pri virtualizácii s vypustením konsolidácie, *spoľahlivosť* a *bezpečnosť* sú základné motivácie pre vznik systémových pieskovísk. Pieskovisko resp. jeho virtualizačná vrstva môže byť vytvorená na rôznej úrovni. Izolácia medzi viacerými virtuálnymi strojmi a medzi VM a prostredím hostiteľa determinuje virtuálne stroje ako efektívnu platformu pre vykonávanie a podporu potenciálne chybových a nedôveryhodných aplikácií. Pri použití technológie virtuálnych strojov je bežnou požiadavkou vykonávať potenciálne nebezpečné operácie vo vytvorenom VM, ktorý je inštanciou prostredia hostiteľa OS. To je možné splniť virtuálnym strojom na úrovni hardvéru. Avšak ako už bolo uvedené, uvedený prístup nie je najefektívnejší, pretože virtuálne stroje sú od seba plne izolované a na každom z nich je spustený samostatný OS. Inicializácia takéhoto stroja vyvoláva vysokú réžiu v otázkach požiadaviek na zdroje a oneskorenia v rámci štartu. V kontexte vzdialenosti virtualizačnej vrstvy od hardvéru je vo väčšine prípadov softvér označujúci sa ako pieskovisko, implementovaný prostredníctvom vrstvy umiestnenej medzi operačným systémom a jednotlivými procesmi. Systémové pieskovisko však môže byť implementované aj ako emulátor alebo virtuálny stroj s vlastným OS. V tejto diplomovej práci je pozornosť venovaná izolovaným prostrediam realizovaným na úrovni operačného systému podľa modelu OS virtualizácie a aplikačnej virtualizácie.

1.3 Analýza existujúcich riešení

Zvolená platforma, na ktorej bude pieskovisko navrhnuté a vytvorené je operačný systém Windows od firmy Microsoft. Alternatívne operačné systémy často obsahujú integrované riešenia pre tento typ problematiky. V unixových OS je možné spomenúť základný nástroj *chroot*, prípadne konkrétne v BSD operačných systémoch program *FreeBSD Jails* [15]. Windows priamo neobsahuje nástroj tohto typu. Dostupné riešenia tretích strán ohľadom tejto kategórie softvéru v súčasnosti pre systém Windows nie sú tak rozšírené a kvalitné na rozdiel od voľne dostupných systémových pieskovísk v unixových systémoch. V nasledujúcich častiach je popísaný princíp, výhody a nevýhody najčastejšie využívaných nástrojov s problematikou pieskovísk. V zozname nie sú uvedené nástroje primárne určené pre systémových administrátorov. Analyzovali sa riešenia prevažne cielené pre domácich a firemných používateľov.

1.3.1 Sandboxie

Najpopulárnejšie komerčné systémové pieskovisko pre OS Windows je program Sandboxie. Inštalátor v priebehu inštalácie nainštaluje do systému potrebný ovládač. Následne je možné pomocou grafického rozhrania (GUI) znázorneného na Obr. 2 spúšťať aplikácie v pieskovisku. Sandboxie implicitne nezakazuje čítanie zo súborového systému. To znamená, že po spustení programu v pieskovisku daný program môže čítať údaje rovnako, ako keby bol spustený mimo pieskoviska. Sandboxie vytvorí na disku adresár (napr. C:\Sandbox), v ktorom uchováva údaje aplikácií spúšťaných v pieskovisku. Príklad: foobar nech je fiktívny program, ktorý si uchováva nastavenia v adresári C:\Users\mato\AppData\Roaming\foobar. Po spustení programu foobar prostredníctvom Sandboxie, sa po novom nastavenia ukladajú do adresára C:\Sandbox\mato\DefaultBox\user\current\AppData\Roaming\foobar. Po odstránení obsahu v tomto adresári sú vymazané všetky zmeny vykonané aplikáciou spúšťanou v pieskovisku. Tieto zmeny je možné vybrať z pieskoviska a uložiť ich natrvalo. Sandboxie podporuje vytváranie profilov pre jednotlivé aplikácie, čím umožňuje nastaviť úroveň bezpečnosti a prípadne definuje, čo všetko sa ma ukladať v pieskovisku resp. mimo pieskoviska. Okrem toho, Sandboxie umožňuje vytváranie viacerých pieskovísk, kde pre každé pieskovisko poskytuje množstvo nastavení ako prístup k internetu, nízkoúrovňový prístup, hardvérový prístup (obmedzenia pre simulovanie klávesnice a myši, spravovanie hardvérových zariadení, meniť sieťové nastavenia) a prístup k súborom a registrom.



Obr. 2 Sandboxie

Sandboxie pozostáva zo softvérového ovládača a klientskej GUI časti, v ktorej pracuje používateľ. Sandboxie je softvér bez voľne dostupného zdrojového kódu, takže vykonanie detailnej analýzy je problémové. Sandboxie podľa autora [16] virtualizuje súbory, diskové zariadenia, registre, objekty procesov a vlákien a objekty pre medzi-procesovú komunikáciu: pomenované rúry, udalosti, mutexy, semafory, sekcie a lokálne volania procedúr. Sandboxie taktiež zabraňuje programom spusteným v pieskovisku explicitné načítanie ovládačov a služieb prostredníctvom centrálneho systému pre správu služieb v systémoch Windows známeho ako Service Control Manager.

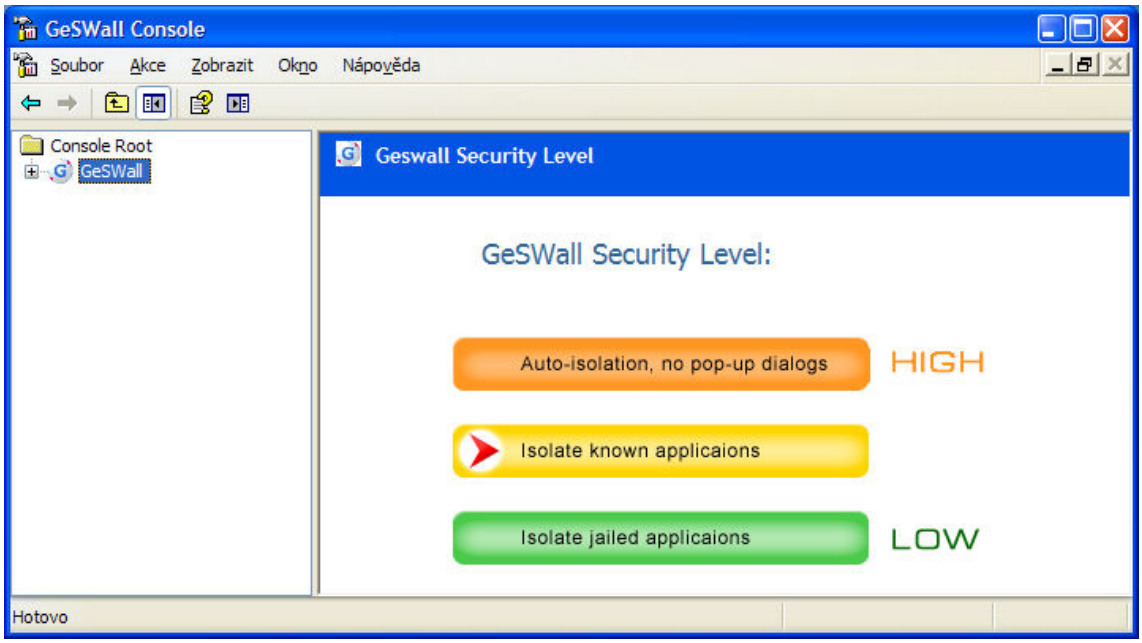
Sandboxie donedávna (júl 2013) plne nepodporoval 64 bitové verzie OS Windows z dôvodu zapnutej ochrany pred modifikáciou jadra OS, detailnejšie popísanej v časti 3.2.1. Toto obmedzenie autori jednoducho obchádzali vypnutím ochrany. Medzi klady aplikácie by som zaradil možnosti pre vytváranie ľubovoľného počtu pieskovísk s nastaviteľnou bezpečnostnou politikou. Ďalej ako jediný nástroj zo všetkých testovaný nevyžadoval po inštalácií reštart systému a bol okamžite použiteľný. Nástroj obsahuje množstvo nastavení s čím na druhej strane súvisí nevýhoda pre nových používateľov v podobe ťažšej konfigurácie. Ďalšie zápory sa týkajú skúsenosťou so stabilitou aplikácie. Po napísaní jednoduchého programu na čítanie a zapisovanie súborov sa mi podarilo Sandboxie zhodiť. V jednom prípade po systémovej aktualizácií Windows 8, Sandboxie prestal fungovať.

1.3.2 GesWall

GesWall na rozdiel od Sandboxie nevyužíva techniku kopírovania pri zápise. Po spustení požadovaného programu v pieskovisku, GesWall pri požiadavke na zápis

nepresmeruje súbor do iného adresára, ale nastaví príznak ACE čím uloží informáciu o tom, že zdrojom modifikácie bol program spustený z izolovaného prostredia. Ak sa iný proces mimo pieskoviska pokúša označený súbor otvoriť, GesWall podľa vopred nastavenej politiky požiadavku buď odmietne alebo po explicitnom odsúhlasení akceptuje. GesWall ako taký nie je pieskovisko v pravom zmysle slova pretože aplikuje obmedzenia aplikácií namiesto ich izolácie do virtuálneho kontajnera.

Po technickej stránke pozostáva GesWall z troch častí: ovládača, spustenej služby na pozadí a klientskej GUI časti zobrazenej na Obr. 3. Ovládač prerušuje systémové volania týkajúce sa registrov a objektov jadra a je implementovaný ako pôvodný filter súborového systému spomenutý v časti 4.1. GesWall po inštalácií a odinštalácií vyžaduje reštart počítača. Služba ovláda stav ovládača, informuje o tom či aplikácia beží v izolovanom prostredí a komunikuje s klientskou časťou.



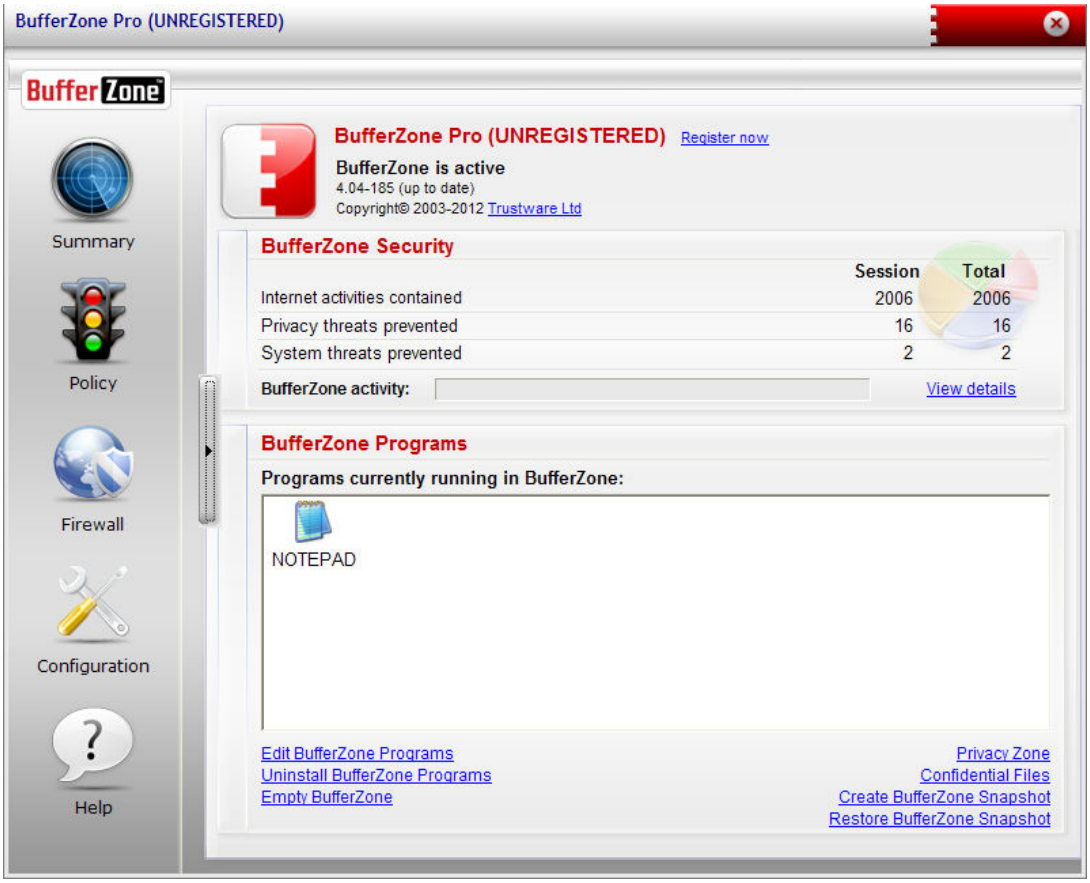
Obr. 3 GesWall

GesWall existuje vo verzii Freeware a Professional. Posledná verzia bola vydaná pred dvoma rokmi pričom podporuje maximálne Windows 7. 64 bitová verzia nebola dokončená a podľa stránok výrobcov sa jedná o mŕtvy projekt. Napriek tomu GesWall pre staršie verzie OS Windows je stále možné použiť, kedy motiváciou a hlavnou výhodou je nízka náročnosť na systémové zdroje. Hlavnou nevýhodou je spomenutý fakt, že program sa ďalej nevyvíja.

1.3.3 BufferZone Pro

BufferZone podobne ako Sandboxie, vytvára na disku adresár s názvom Virtual kam presmeruje všetky zmeny vykonané procesmi spustenými v pieskovisku resp. v „BufferZone“ ako ho označuje výrobca. Okrem toho, v pôvodnej ceste vytvorí nástroj odkaz na súbor ležiaci v pieskovisku. BufferZone vyjmúc súborov podporuje presmerovanie zmien vykonaných v registroch, kontrolu prístupu k internetu a systémovým komponentom.

BufferZone je hlboko integrovaný do systému Windows. Prostredníctvom ovládača kontroluje a prerušuje systémové volania. S používateľom komunikuje GUI aplikácia zobrazená na Obr. 4.



Obr. 4 BufferZone Pro

Aplikácia aktuálne nepodporuje Windows 8 a Windows 8.1. BufferZone existuje vo verzii pre podniky a domácnosti, ktorá je poskytovaná ako freeware. BufferZone sa snaží pôsobiť ako komplexný doplnkový softvér k antivírusovým programom. S tým môže súvisieť jeho výkonový deficit oproti ostatným riešeniam. Silná integrácia do systému Windows a predvolené nastavenia pre internetové prehliadače a ďalší softvér spôsobujú niekedy nechcenú izoláciu a problémy so stabilitou.

1.3.4 Shadow Defender

Shadow Defender rozumie pod pojmom pieskovisko celý disk resp. vybranú partíciu v stave tzv. tieňového režimu. Všetky zmeny vykonané na disku v tieňovom móde sú po reštarte systému Windows odstránené. Počas tohto módu sa používateľovi javí, že súbory sú zapisované ako pri normálnom režime. Po vypnutí tieňového módu a reštarte OS už nové súbory neexistujú, zmazané súbory sú obnovené a zmenené položky majú taký obsah, aký bol pred vstupom do tieňového módu. Shadow Defender môže podľa zvolených nastavení využívať operačnú pamäť ako vyrovnávaciu pamäť vopred definovanej veľkosti, do ktorej sa ukladajú všetky vykonané zmeny. Po vyčerpaní tejto pamäte sú ďalšie zmeny presmerované na disk. O presmerovanie sa stará ovládač na úrovni jadra implementovaný ako súborový filter. Spustenie tieňového režimu sa vykonáva pomocou grafického rozhrania aplikácie.

Shadow Defender je možné používať 30 dní. Následne je potrebné zakúpiť licenciu. Hlavnými výhodami aplikácie je schopnosť transparentne obnoviť stav systému do bodu pred vykonaním zmien a podpora najnovších verzií OS Windows vrátane 64 bitových edícií. Nevýhodou je nutný reštart systému pred každou obnovou a taktiež nemožnosť jednoducho zistiť, ktoré súbory boli počas tieňového režimu vytvorené, modifikované alebo odstránené.

1.3.5 Cameyo

Okrem bližšie popísaných nástrojov existujú riešenia, ktorých primárnym účelom nie je prevencia pred škodlivým softvérom. Do výberu analyzovaných existujúcich riešení je zaradený aj program Cameyo, plnohodnotný zástupca aplikačnej virtualizácie. Cameyo si za prvotný cieľ vybral prenositeľnosť aplikácií, a preto sa nešpecializuje na bezpečnosť. Hlavná myšlienka pozostáva z vytvárania izolovaných aplikácií vo forme jedného spustiteľného súboru prostredníctvom editora balíčkov. Vytváranie takéhoto balíčka je časovo náročné. Preto Cameyo obsahuje nástroj pre sledovanie zmien vykonaných počas inštalácie požadovaného programu. Cameyo si v spoločnom súbore uchováva všetky potrebné súbory a registrové kľúče. Stránka výrobcu obsahuje niekoľko vopred pripravených balíkov s populárnym softvérom. Implementačne je Cameyo riešený technikou prerušení Windows API volaní na používateľskej úrovni, keďže aplikácie zbalené pomocou Cameya nevyžadujú administrátorské oprávnenia. Táto metóda prerušení je detailnejšie popísaná v časti 3.1.

Nevýhodou v použití Cameyo a podobných riešení v súvislosti s bezpečnostným aspektom pieskovísk je prvotné vytváranie balíčkov. Buď využijeme jednoduchšie riešenie využívajúce metódu sledovania zmien vykonaných počas inštalácie s rizikom infikovania systému alebo manuálne, náročnejšie vytváranie balíkov. Ďalšou nevýhodou ohľadom tejto témy je spomenutá implementovaná technika prerušení, ktorá sa dá bez väčších ťažkostí prekonať. Napriek tomu, myšlienka kontrolovaného vykonávania je vo forme jedného súborového kontajneru zaujímavá.

1.3.6 Zhrnutie existujúcich riešení

Všetky analyzované riešenia určitým spôsobom izolovali aplikácie. Využívali široké spektrum techník počnúc kopírovaním modifikovaných súborov na odlišné miesto, značením pozmenených súborov, virtualizáciou celého diskového oddielu alebo použili izoláciu vytvorením jedného samo spustiteľného súboru obsahujúceho samotný program a všetky údaje.

Väčšinu existujúcich riešením trápil problém s pomalou podporou nových verzií OS a 64 bitových edícií, čo de facto súvisí s implementačnou technológiou ovládača. Prerušenia systémových volaní neraz využívajú nezdokumentované funkcie, ktoré sa s každou verziou alebo bezpečnostnou záplatou môžu meniť. Absencia 64 bitových riešení je prevažne spôsobená prítomnosťou ochrany pred modifikáciou kritických častí OS.

Takmer všetky existujúce nástroje zastupujú komerčný softvér. Zadarmo je možné získať požadovaný produkt s licenciu shareware, trialware alebo v lepšom prípade freeware. Prakticky neexistuje riešenie s otvoreným zdrojovým kódom, čo pre niektorých používateľov a inštitúcie predstavuje problém.

Analýza vykonanej činnosti softvéru spusteného v izolovanom prostredí v súčasných riešeniach absentuje. Často chýba názorná ilustrácia alebo aspoň prehľadný textový záznam o činnosti.

V navrhovanom pieskovisku je braný ohľad na všetky spomínané nedostatky. V ďalších častiach diplomovej práce sú načrtnuté súčasné princípy a použité technológie pri implementácií riešení tohto typu v systémoch Windows s následným výberom najvhodnejšej realizácie.

2 Základné princípy riešenia bezpečnostnej politiky na báze systémových pieskovísk

Systémové pieskovisko musí kontrolovať prístup ku komponentom OS. Komponenty môžu tvoriť súbory, registre a iné objekty operačného systému. V tejto kapitole sú popísané základné komponenty a rôzne metódy pri ich ošetrení v nadväznosti na základný návrh riešenia.

2.1 Prístup k súborom

Východisková architektúra navrhovaného riešenia, ako bolo spomenuté v časti 1.2, je jadro pieskoviska vytvorené ako vrstva medzi operačným systémom Windows a jednotlivými aplikáciami. Základná myšlienka podpory virtualizácie na úrovni OS je súborová virtualizácia resp. presmerovanie a obmedzenie žiadostí o prístup k súborom z virtuálneho stroja do lokálnej, koreňovej partície hostiteľského OS. Napr. ak sa proces vo virtuálnom stroji V1 pokúša pristúpiť k súboru s cestou *C:\foo\bar*, virtualizačná vrstva môže presmerovať požiadavku do súboru *C:\V1\foo\bar*. Keď proces v inom prostredí (povedzme V2) chce pristúpiť k *C:\foo\bar*, cesta môže byť presmerovaná na súbor *C:\V2\foo\bar*, ktorý je odlišný od súboru *\foo\bar* vo *V1*. Takéto zobrazenie je vykonávané transparentne vo virtualizačnej vrstve. Proces pristupujúci k súboru *C:\foo\bar* nevie zistiť, že v skutočnosti pristupuje k inému súboru.

Cieľ požiadaviek na súborový systém je možné presmerovať na adresár predstavujúci koreň súborového systému, ktorý bude obsahovať ekvivalentnú stromovú štruktúru alebo do súborového kontajneru. Takýto kontajner sa bude javiť hostiteľskému systému ako jeden súbor. Jadro pieskoviska vedomé si špecifikácie kontajneru môže pristupovať ku všetkým položkám vo vnútri kontajneru a transparentne pre virtualizovaný proces riadiť súborové požiadavky. Implementačne môže mať súborový kontajner formát jednoduchého archívu (napr. ZIP) alebo môže obsahovať kompletný, samostatný súborový systém a využiť existujúce špecifikácie VHD, VMDK a podobne [17].

Relevantným aspektom pri súborovej virtualizácii je fakt či aplikácia vo virtuálnom stroji „vidí“ ostatné súbory a diskové oddiely v hostiteľskom prostredí. V rámci toho môžeme determinovať dva druhy riešení,

- proces nepozná obsah hostiteľského súborového systému,

- proces v rámci jeho nastavenej bezpečnostnej politiky môže čítať ľubovoľný súbor v hostiteľskom systéme.

V prvom prípade dochádza k plnej zmene koreňového adresára rovnako ako pri unixovom nástroji *chroot*. Takýto prístup má výhody vo forme absolútnej súborovej izolácie a menšej výkonnostnej réžie. Na druhej strane, vo virtuálnom koreňovom adresári sa musia nachádzať všetky knižnice a súbory potrebné pre bezproblémové spustenie a beh aplikácií v pieskovisku. To na systémoch Windows môže predstavovať problém, pretože nie je jednoduché zistiť a odhaliť všetky DLL závislosti. Ďalšie ťažkosti zahrňujú načítanie duplicitných DLL knižníc v operačnej pamäti a plytvanie diskovým priestorom. Druhý typ riešenia povoľuje virtualizovanému procesu čítať ľubovoľný súbor v hostiteľskom systéme. Presmerovanie požiadavky spôsobí, že každý súbor ku ktorému proces pristupuje, sa skopíruje do presmerovaného adresára. Avšak duplikovanie zdrojov zvyčajne stojí určitý výkon, zaberá miesto v primárnej pamäti a súčasne zvyšuje réžiu na inicializáciu a ukončenie pieskoviska. Preto sa virtuálny stroj delí o väčšinu zdrojov s hostiteľom a vytvára privátne kópie súborov vo virtualizovanom adresári iba v prípade, kedy to je nutné – pri zápise a modifikácií súborov. Takýto prístup sa vo všeobecnosti nazýva mechanizmom *kopírovania pri zápise* (z angl. copy on write) [18]. Pri metóde kopírovanie súboru pri zápise však treba brať na zreteľ dôležitú skutočnosť. Súbory sa budú vyskytovať na dvoch miestach: v hostiteľskom prostredí a v prostredí pieskoviska. Keď proces vo VM požiada o obsah konkrétneho adresára, potom by mal ako odpoveď dostať zjednotenie oboch umiestnení s vynechaním duplicitných položiek v hostiteľskom adresári.

Odstránenie alebo premenovanie súborov iniciované procesom v pieskovisku by nemalo mať vplyv na hostiteľské prostredie. To znamená, že ak sa aplikácia v pieskovisku pokúsi odstrániť/premenovať súbor nachádzajúci sa mimo pieskoviska, virtualizačná vrstva tento súbor reálne neodstráni. Proces v pieskovisku nebude vedieť, že súbor sa v skutočnosti neodstránil. Virtualizačná vrstva v prípade požiadavky na prístup k vymazanému súboru od tohto istého procesu v konkrétnom pieskovisku oznámi procesu, že daný súbor neexistuje.

2.2 Prístup k registrom

Škodlivý softvér okrem modifikácie súborov zvyčajne zasahuje aj do registrov systému Windows [19]. Registre sú dôležitou súčasťou rodiny operačných systémov firmy Microsoft od vydania Windows 95. Registre sú tvorené hierarchickou databázou

uchovávajúcou systémové informácie a nastavenia programov. Registre pozostávajú z kľúčov a hodnôt. Kľúče sú analogické k súborovým adresárom. Každý kľúč môže obsahovať ďalší kľúč alebo hodnotu. Hodnota predstavuje reťazec, číslo alebo binárne údaje. Na rozdiel od súborov, k hodnotám nie je možné pristupovať priamo, ale iba prostredníctvom kľúča vďaka čomu je napríklad možné pomenovať hodnotu v kľúči rovnako ako kľúč potomka.

V registroch sa okrem iného nachádza konfigurácia operačného systému vrátane kľúčov pre automatické spúšťanie programov [20]. Väčšina programov zapisuje do registrov minimálne počas inštalácie. Pre všetky tieto dôvody sú registre kritickou súčasťou systému a prístup k nim musí byť jadrom pieskoviska patrične ošetrený. To znamená, že pieskovisko odchytí systémové volanie v rámci registrov a modifikuje potrebné kľúče a hodnoty. Princíp môže byť rovnaký ako u súboroch alebo mierne modifikovaný s menšou réžiou, kedy sa celý strom kľúčov a hodnôt priamo prenesie do pieskoviska.

2.3 Komunikácia a synchronizácia medzi procesmi

Procesy v operačnom systéme Windows môžu využívať rôzne metódy pre komunikáciu a vzájomnú synchronizáciu. Tieto metódy reprezentujú rôzne pomenované objekty. Windows poskytuje množstvo synchronizačných objektov, medzi ktoré patria: mutex, semafor, udalosti a časovač. Tieto objekty sú podobne ako súborový systém a registre usporiadané do stromu [21]. Jednotlivé procesy medzi sebou ďalej môžu komunikovať prostredníctvom pomenovaných rúr, mailslotov, synchronizačných objektov, spoločnej pamäte, lokálneho volania procedúr (LPC), schránky, Windows soketov a podobne [22].

Ošetrenie komunikácie a synchronizácie medzi procesmi na rozdiel od súborovej a registrovej virtualizácie u niektorých riešení absentuje, pretože mnoho programov tieto techniky nevyužíva. V záujme najsilnejšej izolácie medzi jednotlivými VM navzájom a medzi VM a hostiteľom je potrebné, aby pieskovisko kontrolovalo a ošetrilo čo najviac spôsobov komunikácie a synchronizácie. Možno rozlíšiť dve rozličné metódy izolácie aplikované na túto problematiku. Prvým spôsobom je ísť cestou reštrikčnej politiky, kedy sa jednoducho zakáže komunikácia medzi procesmi. Niektoré aplikácie k svojej činnosti vyžadujú komunikáciu s inými procesmi a úplný zákaz komunikácie zapríčiní ich nesprávne vykonávanie. Iná technika zahrňuje virtualizáciu menného priestoru pre pomenované objekty. Mená objektov sú v operačnom systéme na

globálnej úrovni. Ich premenovaním do lokálneho priestoru by došlo k izolácií podobne ako pri súborovej virtualizácii.

2.4 Sieťová izolácia

Sieťová izolácia v širšom význame znamená vytvorenie nezávislého prostredia pre ľubovoľný proces komunikujúci po lokálnej sieti a internete [23]. Takýto proces vystupuje pod unikátnou virtuálnou IP adresou v systéme. Podpora takejto izolácie v systémových pieskoviskách nie je rozšírená. Najčastejšie sa vyskytuje pri riešeniach s úplnou implementáciou virtualizácie na úrovni OS, kedy nie je žiaduce, aby jednotlivé virtuálne stroje vystupovali pod rovnakou IP adresou a vzájomne si zasahovali do sieťovej prevádzky. Príkladom použitia môže byť viacero webových serverov bežiacich v rôznych virtuálnych strojoch. Iný prístup, kedy sieťový podsystém v operačnom systéme je emulovaný pre každý proces, je vynikajúcim spôsobom na analýzu chovania škodlivého softvéru. Implementácia takéhoto mechanizmu je komplikovaný a náročný proces, a preto sa vyskytuje hlavne pri hardvérovej virtualizácii, kedy vo virtuálnom stroji beží samostatný operačný systém.

Sieťová izolácia v užšom význame zahrňuje kontrolu a obmedzenie prístupu pre komunikáciu po lokálnej sieti a v internete. Je žiaduce, aby pieskovisko najmä v situácií kedy program ma prístup k hostiteľským súborom, bolo schopné blokovať prístup na internet a zamedziť odoslanie citlivých údajov. Správne nastavený firewall túto požiadavku dokáže jednoducho a efektívne splniť, avšak mnoho bezpečnostných produktov sa spolieha na vlastné riešenia a blokovanie vykonávajú implicitne.

2.5 Kontrola systémových zdrojov

Kontrola systémových zdrojov predstavuje ochranu pred zahltením a zneprístupnením zdrojov počítača. Pieskovisko môže kontrolovať využitie súborového systému, registrov, operačnej pamäte a zaťaženia CPU. V závislosti na nastaveniach následne aplikuje obmedzenia.

Nakoniec, vedľa manažovania zdrojov pieskovisko môže obsahovať politiku v rámci práv systému ako modifikácia systémových nastavení (ikony, písma, nastavenia obrazovky, ovládanie napájania atď.), odhlasovanie, reštart a vypínanie systému.

3 Modifikácia programu spusteného v pieskovisku

Vykonávanie programov na počítačovom systéme s operačným systémom je zvyčajne rozdelené na dve časti. Zatiaľ čo štandardné aplikácie ako tabuľkový procesor alebo editor obrázkov sú vykonávané v režime používateľa, prevažná časť operačného systému je vykonávaná v režime jadra. Táto separácia bráni používateľským programom bezprostredne reagovať so systémom. Napr. používateľský proces nemôže priamo otvoriť súbor. Namiesto toho, OS poskytuje špeciálne rozhranie – *systémové volania*. Systémové volania sú aplikáciám k dispozícií pomocou Windows API rozhrania. Medzivrstva medzi systémovými volaniami a Windows API sa nazýva natívne API. Natívne API tvorí nezdokumentované rozhranie v režime používateľa primárne určené pre internú implementáciu Windows API. Napriek tomu, niektoré používateľské aplikácie môžu priamo pristupovať k natívnemu API.

Aby bolo možné modifikovať správanie ľubovoľného programu spusteného v pieskovisku, potrebujeme buď zdrojové kódy aplikácie (nemáme k dispozícií), modifikovať binárne súbory alebo prerušiť vykonávanie programu na rôznych úrovniach OS. Prerušenia môžu byť zachytené na úrovni jadra alebo na úrovni používateľa. Schopnosť previazať svoj vlastný kód s cudzím kódom (zavesiť sa naň) sa označuje anglickým termínom *hooking*.

3.1 Prerušenia na úrovni používateľa

Body prerušenia na úrovni používateľa predstavujú funkcie Windows API implementované v systémových knižniciach ako napr. kernel32.dll a user32.dll alebo nezdokumentované natívne API funkcie v knižnici ntdll.dll. Aby bolo možné prerušiť volanie, je potrebné modifikovať adresný priestor cieľového procesu. Operačné systémy rodiny Windows NT sú založené na architektúre, kde jeden proces nemôže pristupovať do adresného priestoru iného procesu. Všetky prerušenia na úrovni používateľa vyžadujú modifikáciu adresného priestoru procesu. Najčastejšie sa na to využíva technika *DLL injection*, pomocou ktorej je možné vykonávať požadovaný kód v kontexte iného procesu. Podstata DLL injection spočíva v načítaní DLL knižnice s dotknutým kódom do adresného priestoru procesu. V nasledujúcich častiach sú stručne popísané metódy pre zachytenie cieľovej API funkcie v OS Windows.

3.1.1 Modifikácia adresy API funkcie uloženej v tabuľke IAT

Spustiteľné súbory v OS Windows využívajú formát Portable Executable (PE). PE formát je dátová štruktúra obsahujúca dátové sekcie, kód programu a ďalšie kontrolné štruktúry. Po spustení ľubovoľného programu je v operačnej pamäti zavedený rovnaký PE formát [24]. Jedna sekcia PE formátu v pamäti je IAT (z angl. Import Address Table) štruktúra slúžiaca pre volanie funkcií z DLL knižníc. IAT je tabuľka adries funkcií, ktoré sú vyplnené zavádzačom OS pri načítaní korešpondujúceho DLL. Prerušenie API funkcie je možné dosiahnuť v prvom rade nájdením tabuľky IAT v adresnom priestore procesu a následnou modifikáciou adresy funkcie v IAT na našu, požadovanú funkciu. Modifikácia adries v IAT dokáže zachytiť volania API vo väčšine prípadov, ale nie v situácií ak sa funkcie nevolajú cez tabuľku IAT, napr. ak proces explicitne načítava dynamické knižnice a získava adresu API funkcie priamo v DLL (použitie funkcií API LoadLibrary a GetProcAddress) [25]. Je však možné zachytiť aj tieto volania. Môžeme previazať svoj vlastný kód s kódom funkcie LoadLibrary a sledovať, kedy sú DLL knižnice dynamicky načítané. Následne je možné dané DLL modifikovať podobne ako funkcie v IAT. Iný prístup ako ošetriť dynamicky načítané funkcie, je možnosť zviazať funkciu GetProcAddress tak, že nebude vracať reálnu adresu API funkcie, ale našu požadovanú funkciu.

Modifikácia tabuľky IAT je najčastejší a implementačne najjednoduchší spôsob zachytávania prerušení. Využívajú ho tvorcovia softvéru z oblasti bezpečnosti, ale aj programátori škodlivého softvéru.

3.1.2 Modifikácia tabuľky EAT cieľového DLL

Tabuľka adries exportovaných funkcií (EAT) v DLL obsahuje adresy exportovaných funkcií daného DLL. Prvým krokom pri aplikácií metódy je nájdenie EAT tabuľky v adresnom priestore procesu. Vo chvíli, keď je tabuľka k dispozícií, nasleduje nájdenie adresy požadovanej funkcie podľa jej názvu. Pre úspešne vykonanie zmeny adresy je nevyhnuté nastaviť časť operačnej pamäti, v ktorej leží hľadaná funkcia na zápis. Nasleduje prepis pôvodnej adresy funkcie na požadovanú rutinu a obnovenie zmenených práv pre prístup do pamäte [26]. Odteraz, keď sa proces pokúsi zistiť pomocou GetProcAddress adresu funkcie, obratom dostane novú, zmenenú adresu. Modifikácia EAT tabuľky nie je veľmi náročná, ale oproti ostatným metódam prakticky nemá žiadne výhody.

3.1.3 Prepísanie kódu v operačnej pamäti

Namiesto modifikácie tabuliek IAT a EAT môžeme priamo manipulovať s načítaným binárnym kódom v operačnej pamäti. Existuje niekoľko metód založených na tejto technike. Jednou z nich je prepísanie adresy funkcie volanej inštrukciou CALL [27]. Táto metóda je však komplikovaná a často dochádza k chybám. Iný, sofistikovanejší spôsob pozostáva z nájdenia adresy originálnej API funkcie a zmeny prvých pár bajtov tejto funkcie na inštrukciu JMP, ktorá presmeruje riadenie programu na požadovanú funkciu.

Modifikácia kódu v operačnej pamäti je náročná technika, ktorá môže často pri chybnej implementácií a nesprávnom nasadení spôsobiť pád resp. v lepšom prípade chybné vykonanie programu. V určitých scenároch vrátane kontrolovaného vykonávania, kedy je potrebné prerušiť iba špecifický proces a zároveň počas behu programu byť permanentne zavesený na funkciu, je prepisovanie kódu efektívne a vhodné.

3.1.4 Použitie Proxy DLL

Ovplyvniť vykonávanie programu sa dá aj technikou DLL presmerovania resp. trójskeho DLL, pri ktorej je premenovaná cieľová DLL knižnica a vytvorená zastupujúca DLL knižnica s rovnakým menom a rovnakou množinou funkcií akú má cieľové DLL [28]. Náhradné DLL spracováva iba pre nás zaujímavé API funkcie. Všetky ostatné funkcie vo vytvorenej DLL knižnici používajú nepodmienený skok na vyvolanie zodpovedajúcej funkcie v cieľovej DLL knižnici.

3.1.5 Použitie existujúcej knižnice

Implementácia prerušení nie je triviálna úloha, a preto vzniklo niekoľko knižníc pre uľahčenie práce. V súčasnosti medzi najobľúbenejšie knižnice pre zachytenie Win32 API volaní patria projekty Detours a EasyHook. Detours je produkt firmy Microsoft prvotne vyvinutý pre interné účely, neskôr uvoľnený pre verejnosť. Princíp fungovania knižnice je znázornený na Obr. 5. Detours nahradí prvých pár inštrukcií cieľovej funkcie (z angl. target function) nepodmieneným skokom na používateľom definovanú funkciu (z angl. detour function). Nahradené inštrukcie vrátane inštrukcie nepodmieneného skoku na zvyšný (nenahradený) kód cieľovej funkcie sú uložené v tzv. trampolíne (z angl. trampoline function). Používateľom definovaná funkcia môže buď nahradiť cieľovú funkciu alebo rozšíriť jej sémantiku vyvolaním cieľovej funkcie ako

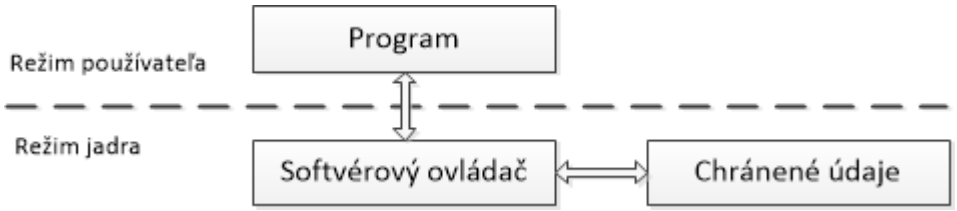
podprogram cez trampolínu [29]. Detours využíva spomínanú techniku prepísania bajtov cieľovej funkcie API. Detours je možné získať zadarmo alebo kúpením licencie. Vo verzii zadarmo chýba podpora pre prerušenie 64 bitových aplikácií a taktiež obsahuje isté obmedzenia.

Obr. 5 Vykonávanie bez prerušenia a s Detours prerušením

EasyHook autor popisuje ako projekt, ktorý „začína tam kde Detours končí“ [30]. EasyHook poskytuje okrem štandardného API pre zachytávanie volaných funkcií, rozšírené API pre prerušenia vykonávané z manažovaného kódu pre nemanažovaný kód a vice versa. Znamená to podporu pre prerušovanie ľubovoľných funkcií napr. v jazyku C#. Knižnicu je možné transparentne použiť v 32 a 64 bitových edíciách Windows od verzie Windows 2000 SP4.

3.2 Prerušenia na úrovni jadra

Prerušenia na úrovni jadra sú v porovnaní s preruseniami na úrovni používateľa spoľahlivejšie, robustnejšie a ťažšie prekonateľné, ale na druhej strane ich je náročnejšie implementovať. Štandardné aplikácie v normálnom režime nemajú prístup k chráneným údajovým štruktúram jadra. Preto sú implementované na úrovni jadra prostredníctvom softvérových ovládačov. Ovládač je softvérový komponent umožňujúci komunikáciu medzi hardvérom a operačným systémom a ďalším softvérom. Softvérový ovládač nie je asociovaný s reálnym zariadením, ale predstavuje modul vykonávaný v prostredí jadra. Program komunikuje so softvérovým ovládačom, ktorý má prístup k chráneným údajom. Schematické zobrazenie softvérového ovládača ilustruje Obr. 6. Môžeme uvažovať o dvoch hlavných prístupoch pri preruseniach na úrovni jadra, ktoré sú detailnejšie popísané v nasledujúcich častiach práce.



Obr. 6 Softvérový ovládač

3.2.1 Modifikácia SSDT tabuľky

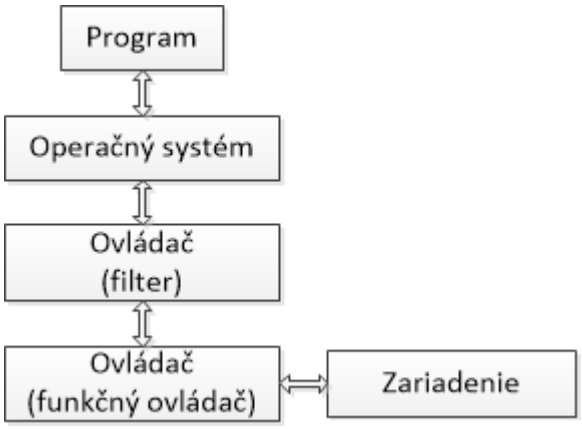
Jednou z metód prerušení systémových volaní na úrovni jadra je nahradenie záznamu v tabuľke obsahujúcej všetky používané systémové služby (z angl. System Service Dispatch Table - SSDT) adresou vlastnej funkcie, ktorá môže volať originálne systémové volanie vrátane vykonania pre a post spracovania [31]. SSDT obsahuje adresy všetkých systémových volaní jednoznačne identifikovaných identifikátorom systémového volania.

Modifikácia SSDT tabuľky predstavuje hrubý zásah do jadra systému Windows a oficiálne by sa nemala používať, pretože ohrozuje bezpečnosť a stabilitu systému. S príchodom prvej 64 bitovej edície Windows, konkrétne Windows XP Professional x64 Edition, sa Microsoft rozhodol implementovať ochranu proti modifikáciám v jadre s názvom PatchGuard. Uvedená ochrana bola aplikovaná iba pre 64 bitové verzie z dôvodu, že pôvodné ovládače museli byť pre 64 bitový systém aj tak prepísané, čo by u 32 bitovej edície pri niektorých systémových aplikáciách spôsobilo problémy s kompatibilitou. PatchGuard pravidelne v intervaloch kontroluje kód jadra a v prípade neoprávnených zmien vyvolá systémovú výnimku, reštartuje systém a obnoví pôvodný stav vrátame SSDT tabuľky. Napriek tomu, väčšina pieskovísk a niekoľko antivírusových riešení využíva túto metódu.

3.2.2 Filter ovládač

Technika založená na architektúre, pri ktorej sú implementované ovládače jadra v rôznych vrstvách. Väčšina systémových volaní aplikácií poslaných pre ovládače zariadení má formát IRP (z angl. I/O Request Packets). IRP spracováva súvisiaca funkcia daného ovládača. IRP je však typicky spracovaný niekoľkými ovládačmi usporiadanými vo forme zásobníka. Každý ovládač v zásobníku je asociovaný s objektom zariadenia reprezentujúci logické, virtuálne alebo fyzické zariadenie, pre ktoré ovládač spravuje vstupno-výstupné (V/V) požiadavky. IRP pri spracovaní v rámci zásobníka zariadení, je ako prvému zaslaný ovládaču na vrchole zásobníka. Tento ovládač sa nazýva filter. Filter zachytáva všetky V/V požiadavky pôvodne smerované

ovládaču v hierarchii pod ním. Predtým ako V/V požiadavky pošle nižšej, základnej vrstve, filter môže uskutočniť vlastné modifikácie alebo ich ihneď poslať ďalej [32]. Schematické umiestnenie filtra v zásobníku ovládačov je zobrazené na Obr. 7.



Obr. 7 Umiestnenie filtra v zásobníku ovládačov

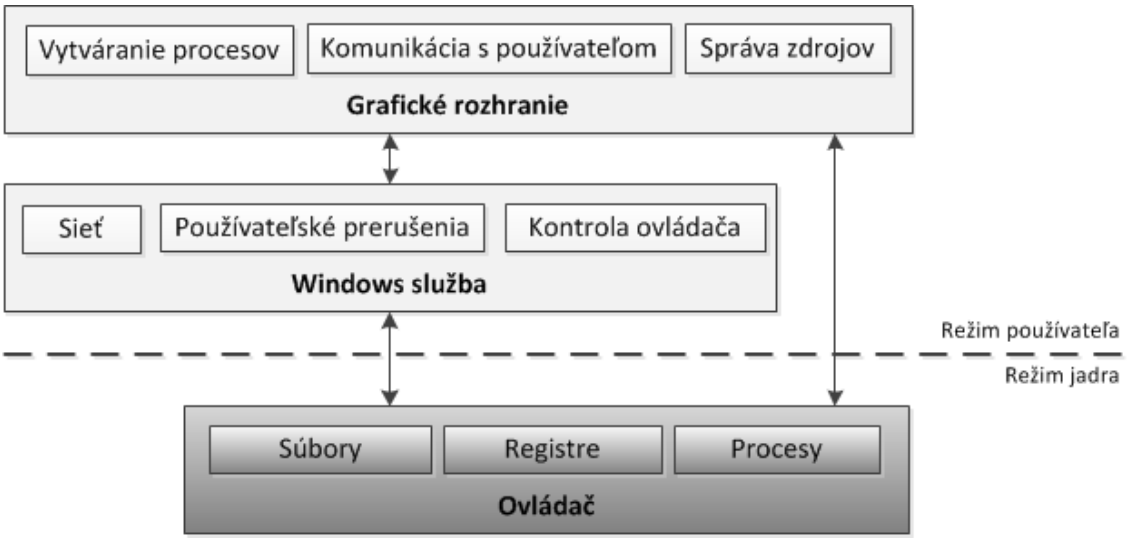
Windows štandardne podporuje filtrovanie ľubovoľného zariadenia a súborového systému. Od verzie Windows XP, Microsoft poskytuje aj základné API pre filtráciu a monitorovanie prístupov do registrov.

3.3 Zhrnutie a výber metódy

Implementácia systémového pieskoviska sa môže uskutočniť pomocou prerušení na úrovni používateľa alebo v privilegovanom režime, na úrovni jadra. Prerušenia v režime používateľa nikdy nemôžu byť metódou pre aplikáciu dodatočných bezpečnostných kontrol v akomkoľvek kontexte bezpečnosti. V prípade izolácie nejakého vyhradeného procesu, ktorý dostatočne poznáme a zároveň tento proces si nie je vedomý existencie virtualizačnej vrstvy, prerušenia na úrovni používateľa môžu postačovať. Skutočný, v praxi použiteľný bezpečnostný softvér sa však nikdy nemôže úplne spoliehať na previazanie vlastného kódu s cudzím kódom iba v režime používateľa.

Navrhované pieskovisko bude pozostávať z troch samostatných častí. Jadro pieskoviska sa bude vykonávať na úrovni jadra. Hlavný modul bude pozostávať zo softvérového ovládača. V novších verziách OS Windows, konkrétne od verzie Vista, Microsoft značne zlepšil použitie filtračného rozhrania. Toto rozhranie predstavuje odporúčaný spôsob pre analýzu a zmenu chovania používateľských programov bez potreby modifikácie dôležitých častí OS. Filtrované budú súbory, registre a procesy. Ovládač bude z väčšej časti implementovaný ako súborový filter. Ostatné prerušenia budú implementované v režime používateľa s využitím knižnice EasyHook. Vloženie

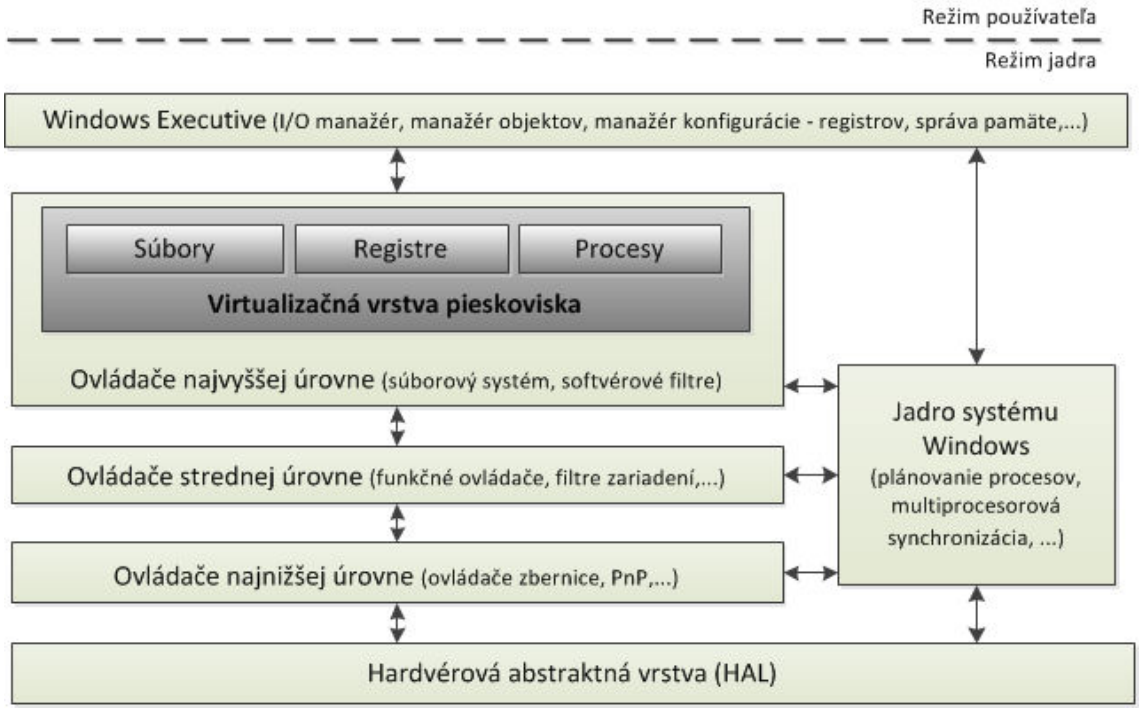
DLL a činnosti vykonávané s oprávneniami administrátora bude vykonávať implementovaná služba systému Windows. Samotný ovládač ani služba nedisponujú grafickým rozhraním. Komunikácia s používateľom sa uskutoční pomocou tretieho samostatného programu. Používateľ bude pieskovisko ovládať prostredníctvom grafického rozhrania, ktoré bude implicitne komunikovať s ovládačom a službou. Základná architektúra pieskoviska je zobrazená na Obr. 8.



Obr. 8 Základná architektúra pieskoviska

4 Jadro pieskoviska

Jadro navrhovanej aplikácie predstavuje virtualizačnú vrstvu, abstrakciu medzi aplikáciami a operačným systémom. Je to modul obsahujúci funkčnosť týkajúcu sa správy súborového prístupu, registrov, procesov a podobne. Umiestnenie virtualizačnej vrstvy z hľadiska architektúry systému Windows zobrazuje Obr. 9. Ovládače sa môžu nachádzať na rôznej úrovni. Jadro pieskoviska pozostáva z ovládača umiestneného na najvyššej úrovni.



Obr. 9 Umiestnenie virtualizačnej vrstvy jadra pieskoviska v systéme Windows

Microsoft poskytuje množstvo techník a softvérových rámcov pre tvorbu ovládačov. Výber konkrétnej technológie závisí od použitia ovládača. Microsoft [33] rozlišuje funkčné ovládače zariadení, filtre zariadení, softvérové ovládače, filtre súborového systému a ovládače súborového systému. Pre vývoj nových funkčných ovládačov sa najčastejšie používa rozhranie KMDF prípadne UMDF pre ovládače nevyžadujúce priamy prístup k štruktúram jadra ako napr. ovládače tlačiarní. Niektoré technológie zase môžu vyžívať tzv. model mini-ovládačov. V tomto modeli ovládač pozostáva z dvoch častí: všeobecnej a špecifickej. Všeobecný diel je rovnaký pre všetky zariadenia. Špecifický je závislý od konkrétneho zariadenia. Zvyčajne vytvára Microsoft všeobecnú časť a výrobca zariadenia špecifickú. Filtre zariadení sa implementujú pomocou KMDF alebo UMDF rozhrania s využitím rôznych technológií

ako Windows platforma pre filtrovanie (WFP), NDIS a podobne. Pri softvérových ovládačoch je možné použiť KMDF alebo „legacy Windows NT“ model. Pri oboch modeloch sa programátor nemusí zaoberať správou napájania a „Plug and Play“ (PnP) mechanizmom. KMDF ošetruje PnP a napájanie automaticky bez potreby zásahu programátora. Windows NT legacy model je od týchto technológií úplne izolovaný. Predchádzajúce tri typy ovládačov je možné vytvoriť aj vo všeobecnom WDM modeli, avšak tvorba takýchto ovládačov je náročnejšia a v súčasnosti by sa mala využívať minimálne. Posledné kategórie ovládačov, filtre súborových systémov a ovládače súborových systémov vyžívajú techniku *minifiltrov* alebo *pôvodný model filtrov*, v angličtine označovaný termínom „legacy filter model“.

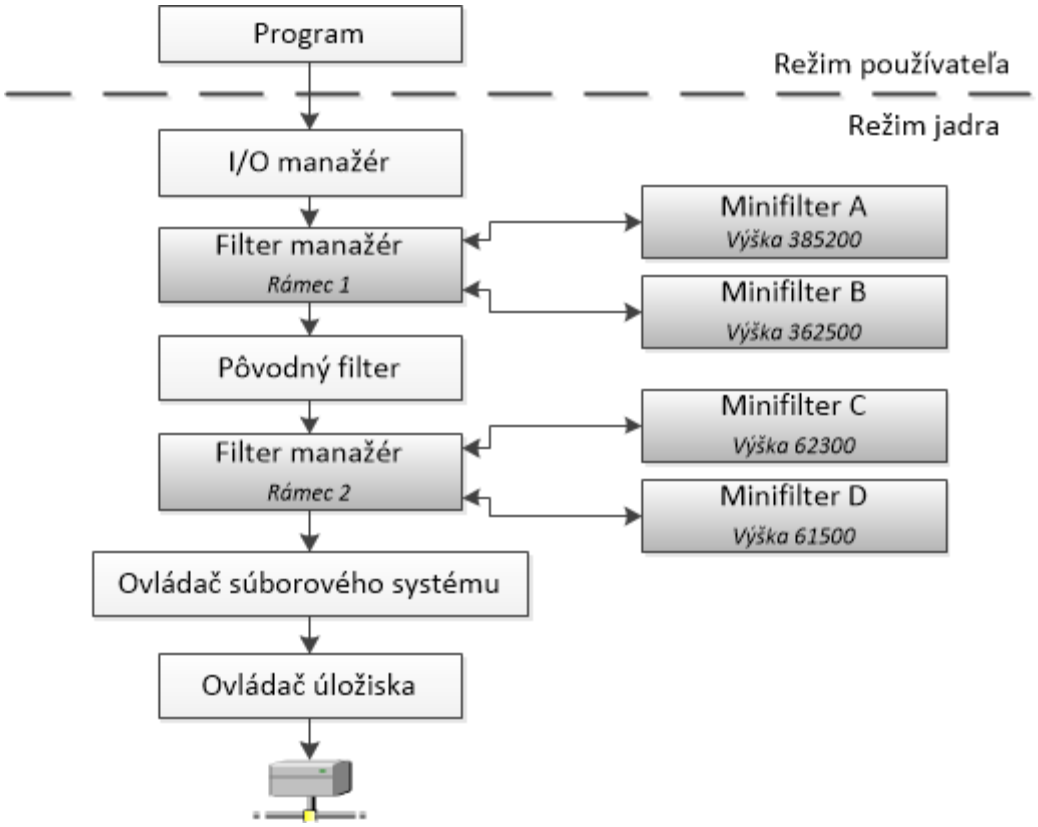
Prvoradý predpoklad na jadro pieskoviska je presmerovanie požiadaviek na súborový systém. Preto bol ako hlavný model ovládača vybraný minifilter model, detailnejšie popísaný v nasledujúcej časti.

4.1 Minifilter

Ovládač, ktorý je vložený medzi Windows NT I/O systém a základný ovládač súborového systému sa nazýva filter súborového systému. Zásobník ovládačov, ako bolo spomenuté v časti 3.2.2, pracuje so štruktúrou IRP. IRP sú v prípade filtrov požiadavky pre vykonanie špecifickej operácie nad súborovým systémom napr. otvorenie, zápis, čítanie, zatvorenie atď. I/O manažér nachádzajúci sa na vrchole zásobníka ovládačov súborového systému akceptuje systémové volania procesov alebo volania z iných komponentov jadra, prekladá súborový deskriptor – identifikátor, do súborových objektov (reprezentuje súbor pri V/V operáciách) a generuje IRP požiadavky, ktoré potom smeruje postupne zhora nadol ovládačom v zásobníku. Potom, čo je IRP požiadavka dokončená, I/O manažér dokončí systémové volanie alebo dokončí inú akciu v prípade asynchrónnej požiadavky.

S príchodom Windows XP SP2, Microsoft vytvoril model pre uľahčenie tvorby filtrov súborových systémov ako alternatívu k predtým štandardne používanému pôvodnému modelu filtrov. Nazval ho model manažéra filtrov. V tomto prípade sú ovládače – minifiltre spravované filter manažérom (FM). FM naprogramoval Microsoft, pričom v skutočnosti je to pôvodný filter ovládač. IRP požiadavky sú najprv spracované FM. Minifilter zaregistruje *spätné volania* (z angl. callback) pre požadované operácie, ktoré následne FM zavolá. FM zapuzdrí IRP do štruktúry označovanej *CallbackData* a postupne, podobne ako pri pôvodných filtroch, smeruje štruktúru jednotlivým

inštanciám minifiltra. Inštancia minifiltra je abstrakcia charakterizujúca zachytenie minifiltra na konkrétnej diskovej partícií. Na rozdiel od pôvodných filtrov je usporiadanie minifiltrov definované pomocou nadmorskej výšky. Čím je nadmorská výška väčšia, tým je minifilter v zásobníku vyššie. Pre dosiahnutie koexistencie medzi pôvodnými filtrami a minifiltrami, FM sa môže zachytiť na viac ako jednom mieste. Každý takýto objekt manažéra filtrov sa nazýva rámec. Z perspektívy pôvodného filtra, je každý rámec len ďalší pôvodný filter. Zjednodušený pohľad na V/V zásobník s dvoma rámcami manažéra filtrov, pôvodným filtrom a inštanciami minifiltra je znázornený na Obr. 10.



Obr. 10 Zjednodušený pohľad na V/V zásobník s inštanciami minifiltra

Spätné volanie minifiltra pozostáva z dvoch samostatných volaní. Minifilter si tak môže zaregistrovať *pre-operáciu*, *post-operáciu* alebo obe operácie [34]. Keď FM spracováva V/V operáciu, volá pre-operáciu každého minifiltra, v poradí podľa nadmorskej výšky, ktorý si zaregistroval dané spätné volanie pre tento typ operácie. Ak minifilter skončí s vykonávaním operácie, vracia vykonávanie naspäť do réžie FM. FM zavolá pre-operáciu s CallbackData štruktúrou nižšieho minifiltra a postup sa opakuje. Keď všetky minifiltre dokončia svoje pre-operácie, FM posiela V/V operáciu pôvodným filtrom a súborovému systému. Post-operácia začína vtedy ak I/O manažér

posúva operáciu do fázy dokončenia. Potom ako V/V operáciu dokončí súborový systém (napr. súbor sa zapíše na disk) a pôvodný filter, k spracovaniu sa dostáva FM. FM volá zaregistrované post-operácie minifiltrov v opačnom poradí ako v pre-operácii t.j. od najnižšej nadmorskej výšky po najvyššiu. Po dokončení všetkých post-operácií FM odovzdáva štruktúru s požadovanými údajmi I/O manažérovi.

Dôležitou súčasťou architektúry filtrov je podpora kontextov. Kontext je štruktúra definovaná minifiltrom pre uchovávanie ľubovoľných údajov asociovaných s objektmi manažéra filtrov. Kontext je možné priradiť inštanciám minifiltra, súborom, súborovým objektom, otvorenému prúdu k súboru (z angl. stream handle context), diskovým oddielom a transakciám.

Model minifiltrov prináša v porovnaní s pôvodnými filtrami viacero výhod. Minifilter je možné počas behu OS dynamicky pripájať a odpájať, FM poskytuje relatívne menej komplikované rozhranie vrátane metód pre získavanie informácií o menách súborov, jednoduchšiu komunikáciu s aplikáciami na úrovni používateľa, efektívnejší manažment kontextov a možnosť filtrovať iba požadované operácie, zatiaľ čo pôvodný filter musí prerušovať všetky operácie. Model minifiltrov napriek týmto výhodám neuľahčuje tvorbu filtrov, čo je dôsledok komplexnosti V/V subsystému, avšak umožňuje eliminovať množstvo kódu redundantného pre každý filter.

4.2 Filtrovanie súborov

Premenovanie cesty k súboru na umiestnenie v doméne pieskoviska je prvým krokom izolácie prístupu z jadra pieskoviska. Nech súkromný priestor (umiestnenie) je ľubovoľný adresár alebo cesta z adresárového stromu predstavujúceho pieskovisko. Napríklad „C:\ESU\Pieskovisko1“ je koreňový adresár pieskoviska s názvom „Pieskovisko1“. Ľubovoľný adresár v tomto umiestnení sa bude označovať pojmom súkromný priestor alebo doména pieskoviska. Štandardný prístup mimo súkromného adresára naopak hostiteľský priestor. Tieňový súbor bude predstavovať obraz hostiteľského súboru v priestore pieskoviska.

V navrhovanom riešení je uplatnený modifikovaný mechanizmus *kopírovania pri zápise*, ktorého základný princíp a výhody boli spomenuté v časti 2.1. Po vytvorení pieskoviska je súkromný priestor prázdny. Proces pristupujúci k existujúcemu súboru nie je presmerovaný do súkromného adresára pokiaľ sa nevyskytne prvá žiadosť na *otvorenie súboru s prístupom pre zápis*. To znamená, že ako náhle proces v pieskovisku

otvorí súbor na zápis, ovládač skopíruje tento súbor do súkromného priestoru. Každý ďalší prístup k súboru sa tak týka jeho kópie. Tento princíp ma výhody vo forme nízkej réžie virtualizačnej vrstvy s čím súvisí jednoduchšia implementácia. Nevýhodou je plytvanie miestom a čas potrebný na vykonanie kópie. Aplikácie častejšie čítajú ako zapisujú súbory, a preto je tento prístup vo väčšine prípadov dostatočný.

4.2.1 Základný princíp pri otváraní a vytváraní súborov

Práca so súbormi je pre koncového používateľa relatívne jednoduchá. Z hľadiska operačného systému je to komplexný proces pozostávajúci z množstva fáz. Všeobecný pohľad pozostáva z otvorenia/vytvorenia súboru (najčastejšie podľa názvu) a získania jednoznačného identifikátora reprezentujúceho súbor v systéme. Následne s využitím identifikátora sa vykonávajú požadované operácie na súbore, napr. čítanie, zápis, získavanie rôznych informácií, premenovanie, odstránenie súboru a podobne. Každá takáto operácie je reprezentovaná (nie vždy) korešpondujúcou IRP operáciou. Poslednou časťou pri práci so súbormi je uvoľnenie identifikátora, často označované ako zatvorenie súboru. Jadro pieskoviska sa primárne zaujíma o prvú fázu, detailnejšie dekomponovanú na nasledujúce pod úlohy. Pre ilustráciu je taktiež uvedené, že používateľ chce otvoriť súbor „C:\foo\bar.pdf“.

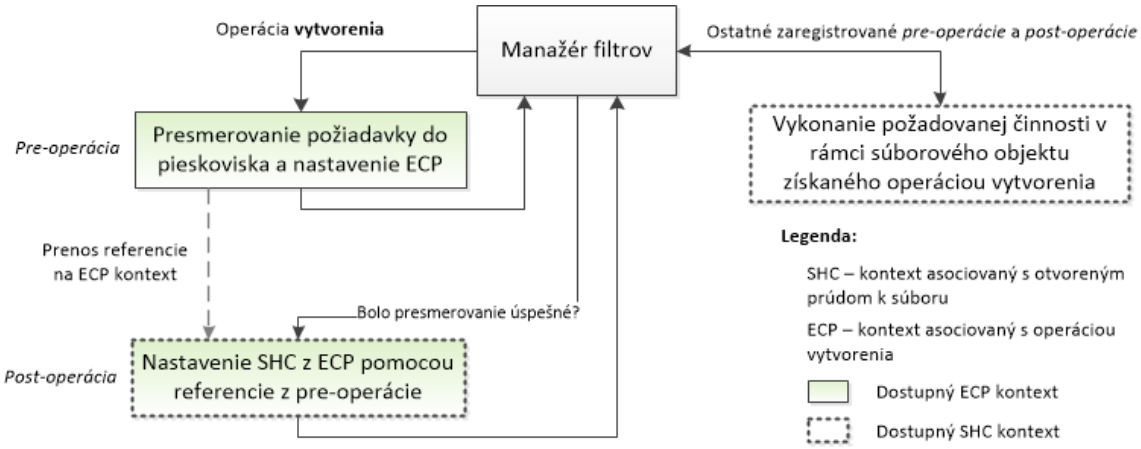
1. Otvorenie súboru v režime používateľa najčastejšie prostredníctvom názvu s využitím funkcie napr. CreateFile dostupnou v aplikačnom programovom rozhraní.
2. Postupné volanie funkcií s parametrami z nižších vrstiev, až sa vykonávanie dostane do režimu jadra a k spracovaniu pristupuje I/O manažér. Požadovaný súbor je v tejto chvíli reprezentovaný tvarom „\??\C:\foo\bar.pdf“.
3. I/O manažér overí parametre funkcií, determinuje cieľový diskový zväzok a iniciuje IRP operáciu vytvorenia s názvom IRP_MJ_CREATE, vrátane alokovania súborového objektu reprezentujúceho súbor. Cesta k súboru má teraz formát „\Device\HarddiskVolume2\foo\bar.pdf“, pretože „\??\C“ ukazuje na zariadenie „\Device\HarddiskVolume2“.
4. IRP je smerovaná k filtrom, ktoré vykonajú potrebné úlohy. V prípade minifiltrov sa toto spätné volanie označuje ako pre-operácia. Forma cesty k uvedenému súboru v rámci súborového objektu je v tomto momente „\foo\bar.pdf“.

5. IRP požiadavka sa dostane k súborovému systému, ktorý vyplní požadované údaje štruktúry podľa kontextu súboru a reálneho zväzku a začne dokončovaciu fázu u pôvodných filtrov, u minifiltrov označovanú ako post-operácia.
6. Filtre dostanú IRP požiadavku a vykonajú požadované post-operácie.
7. I/O manažér podľa výsledného stavu IRP a dát od filtrov môže vytvoriť novú požiadavku a starú eliminovať alebo vrátiť výsledný stav vrátane nového deskriptora k súborovému objektu volajúcemu procesu.
8. Používateľ dostane súborový deskriptor k súboru.

Popísaný princíp predstavuje zjednodušený abstraktný pohľad na operáciu a nezahrňuje sprevádzané udalosti ako použitie vyrovnávajúcej pamäte, asynchrónne spracovanie a množstvo vedľajších faktorov.

Modifikáciu cesty v reťazci operácie vytvorenia vykonáva jadro pieskoviska v rámci 4. kroku. Pôvodná, nevyhovujúca požiadavka nikdy nedosiahne úroveň súborového systému. Elementárny mechanizmus umožňujúci zmenu názvu súboru je založený na nastavení stavu **opätovnej analýzy** pre korešpondujúcu IRP operáciu v pre-operácii. Tento stav signalizuje, že I/O manažér „zahodí“ aktuálne IRP a iniciuje nové IRP so súborovým objektom obsahujúc umiestnenie súboru v doméne pieskoviska. Okrem zmeny cesty a stavu IRP operácie, musí jadro **dokončiť aktuálnu operáciu**, aby sa IRP štruktúra nedostala k súborovému systému. To sa dosiahne vrátením vhodného stavu z pre-operácie manažérovi filtrov. Pri tomto spôsobe presmerovania operácie nastáva dôležitá otázka. Ako filter zistí, ktorá požiadavka už bola presmerovaná a ktorá nie? S príchodom Windows Vista je možné ku každej operácii vytvorenia pridať tzv. ECP kontext. Tento kontext sa zachováva aj pri dokončení IRP, kedy I/O manažér pripája referenciu na kontext k novej IRP štruktúre. Jadro využíva ECP kontext na označenie presmerovanej operácie vo fáze pre-operácie a označenie operácie vytvorenia iniciovanej pieskoviskom v rámci presmerovania medzi diskovými oddielmi v celom ovládači. ECP kontext je dostupný iba v pre-operácii a post-operácii vytvorenia. Pre uchovávanie páru ciest tvoreného z cesty v rámci domény pieskoviska a hostiteľského umiestnenia, pre jednoduchú identifikáciu požiadavky a ďalšie potrebné údaje, je ECP kontext transformovaný na kontext asociovaný s otvoreným prúdom k súboru (SHC) v rámci post-operácie vytvorenia, kedy na rozdiel od pre-operácie vytvorenia súborový objekt už existuje, a preto mu je možné priradiť kontext asociovaný s otvoreným

prúdom k súboru. Tento kontext jadro asociuje iba so súborovým objektom reprezentujúci súbor v pieskovisku a je dostupný iba pre náš minifilter na rozdiel od ECP kontextu dostupného pre všetky ovládače zapojené do spracovania operácie vytvorenia. Platnosť kontextov znázorňuje Obr. 11.



Obr. 11 Platnosť kontextov v jadre pieskoviska

4.2.2 Metóda rozhodovania pri presmerovaní

Navrhnuté jadro pieskoviska nefiltruje všetky typy súborov, ktoré sa môžu objaviť pri otváraní a vytváraní súborov. Ignorované sú požiadavky na

- otvorenie súborov súvisiacich so stránkovaním,
- súbory otvorené prostredníctvom súborového ID,
- súbory týkajúce sa technológie Prefetch.

Jedná otázka vyplývajúca z metódy kopírovania pri zápise nastáva ak proces v pieskovisku otvára súbor. Ako správne jadro vie či súbor má byť pristupovaný z hostiteľského prostredia alebo privátna kópia už bola vytvorená a súbor tak má byť pristupovaný z domény pieskoviska? Priamočiarou metódou by bolo udržiavanie zoznamu mien súborov a adresárov, ktoré už boli skopírované v pamäti. Inou metódu je neustála kontrola existencie súborov. V ovládači je implementovaný druhý spôsob, ktorý pri intenzívnej V/V aplikácií spôsobí výkonnostný prepad avšak je pamäťovo nenáročný, čo je pre kód vykonávajúci sa v prostredí jadra veľmi dôležitý aspekt. Okrem toho je možné voliteľne, počas kompilácie ovládača aktivovať hybridný režim oboch techník, implementovaný ako rýchla vyrovnávajúca pamäť, ktorá si uchováva informácie o menách súborov (počet vzhľadom k veľkosti pamäti a jedinečnosti hašovacej funkcie), ktoré sa nenachádzajú v doméne pieskoviska. Aplikácie mnohokrát

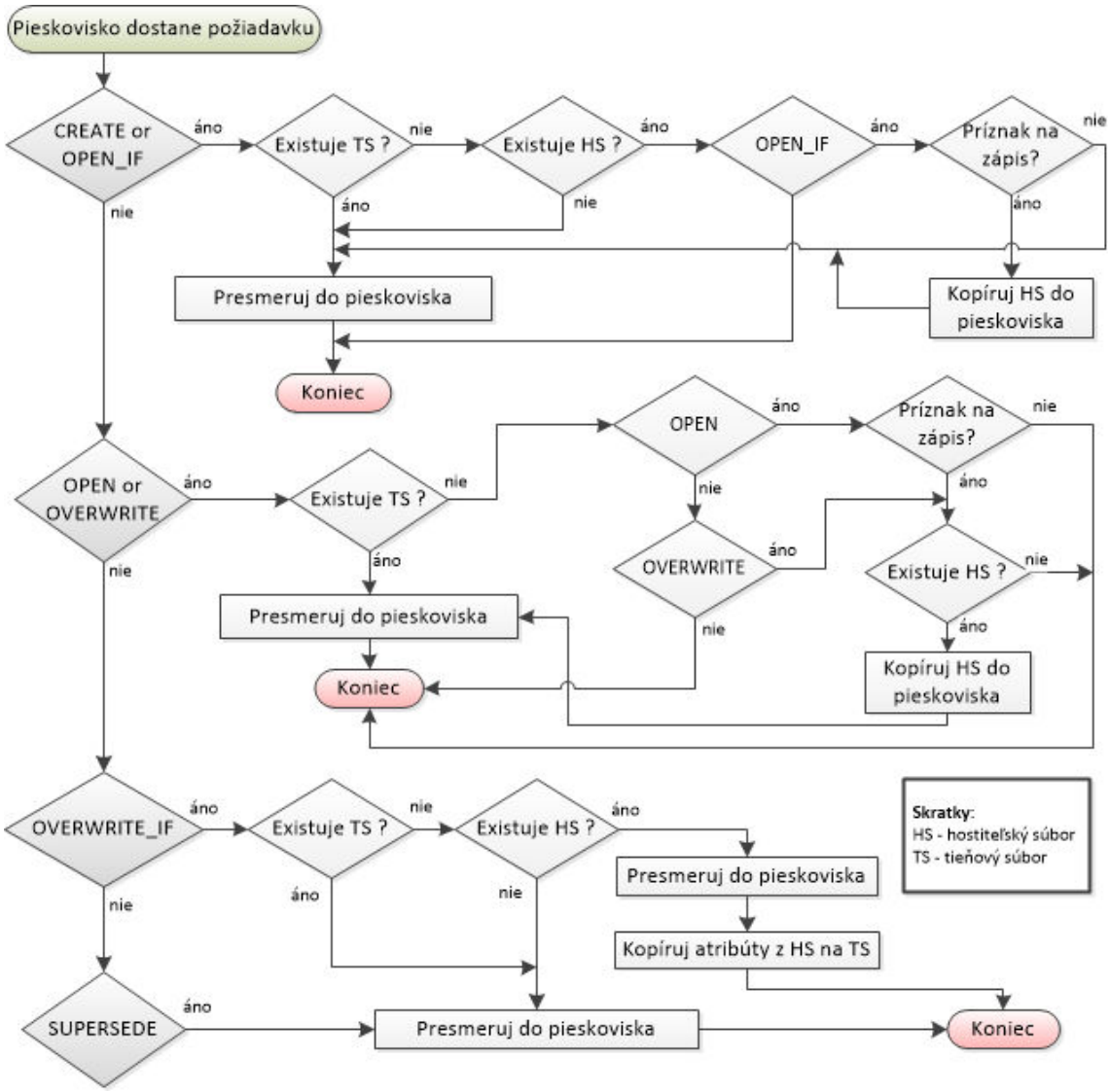
iniciujú požiadavky k rovnakým súborom, čím je pomocou tejto pamäte možné ušetriť niekoľko V/V požiadaviek a zrýchliť najmä štart procesov.

Každá aplikácia pri otvorení/vytvorení súboru špecifikuje ako sa má OS správať ak súbor existuje resp. neexistuje. Systém Windows rozlišuje *šesť dispozícií* prehľadne zapísaných v Tab. 1.

Tab. 1 Dispozície pri otváraní súboru

Dispozícia	**Súbor existuje**	**Súbor neexistuje**
CREATE	požiadavka zlyhá	vytvorený
OPEN_IF	otvorený	vytvorený
OPEN	otvorený	požiadavka zlyhá
OVERWRITE	otvorený a prepísaný	požiadavka zlyhá
SUPERSEDE	nahradený	vytvorený
OVERWRITE_IF	otvorený a prepísaný	vytvorený

Program pri práci so súbormi taktiež špecifikuje *požadovaný prístup*: čítanie, zápis, možnosť odstrániť súbor atď. Špecifikácie dispozície a žiadaného prístupu sú hlavnými elementmi pri rozhodovaní, kedy má jadro presmerovať požiadavku, kopírovať súbor do domény pieskoviska alebo ignorovať požiadavku. Napríklad, aplikácia chce otvoriť ľubovoľný súbor s možnosťou zápisu, pričom ak súbor neexistuje, má byť vytvorený. Počas otvárania súboru program špecifikuje dispozíciu OPEN_IF a príznak na zápis. Jadro pieskoviska dostane požiadavku a determinuje zadanú dispozíciu. Následne ak existuje tieňový súbor, požiadavka sa presmeruje do domény pieskoviska. Ak tieňový súbor neexistuje, rozlišujú sa tri prípady. Ak hostiteľský súbor existuje a operácia má nastavený príznak na zápis, súbor sa skopíruje do pieskoviska a presmeruje sa naň požiadavka. Ak hostiteľský súbor existuje a požiadavka nemá nastavený príznak na zápis, požiadavka sa ignoruje a posiela ďalej. Posledná situácia nastáva vtedy, keď hostiteľský súbor neexistuje. V tomto prípade sa požiadavka implicitne presmeruje do pieskoviska. Všetky ostatné situácie, ktoré môžu nastať v nadväznosti na zadanú dispozíciu a zvolený príznak ilustruje vývojový diagram na Obr. 12. V diagrame nie je kvôli prehľadnosti zahrnutá logika zahrňujúca manažment vymazaných položiek, detailne popísaná v nasledujúcej časti.



Obr. 12 Tok požiadavky v závislosti od dispozície a prístupu

Ďalšia otázka môže nastať v situácií, kedy izolovaný proces iniciuje požiadavku obsahujúcu cestu priamo na súbor v pieskovisku. Takáto požiadavka nie je korektná a jadro pieskoviska ju zamietne.

Navrhnutý filter musí zvládať presmerovania medzi diskovými zväzkami. Koľko zväzkov je v systéme, toľko inštancií minifiltra tvorí jadro pieskoviska. Napr. v prípade, že proces chce zapisovať na zväzok označený písmenom D, a doména pieskoviska sa nachádza na zväzku C, všetka réžia (zisťovanie existencie súboru, otváranie potrebných súborov a pod.) potrebná pri metóde rozhodovania na rozdiel od operácie v rámci jedného zväzku, prechádza cez celý V/V zásobník. Aby takáto interná operácie nebola presmerovaná, ku každej operácii medzi zväzkami je pripojený ECP kontext.

4.2.3 Odstránené súbory

Škodlivý softvér sa často pokúša odstrániť dôležité súbory. Prevencia proti takémuto správaniu musí byť preto vo virtualizačnej vrstve náležite implementovaná. Počas vykonávania programu môže nastať situácia, kedy bol súbor skopírovaný do domény pieskoviska a následne tento istý program alebo iný program v pieskovisku ho odstránil. V tomto prípade sa súbor môže stále nachádzať u hostiteľa, ale nie v súkromnom priestore. Pre izolovaný proces sa to môže javiť ako keby súbor nikdy nebol do domény pieskoviska skopírovaný, pričom chybne podľa základnej logiky, presmeruje požiadavku na existujúci hostiteľský súbor. Je samozrejmé, že pieskovisko nesmie nikdy odstrániť súbor z hostiteľského priestoru. Navrhnuté jadro preto tento problém rieši prostredníctvom pamäťovej štruktúry uchovávajúcej zoznam všetkých súborov, ktoré sa ľubovoľný proces v pieskovisku pokúsil odstrániť. Reálne sa zo súborového systému súbor odstráni iba ak sa nachádza v doméne pieskoviska.

Operačný systém Windows poskytuje dva spôsoby pre odstránenie súborov. Menej štandardný mechanizmus pozostáva z nastavenia špeciálneho príznaku počas otvárania resp. vytvárania súboru. Po zavretí súboru je súbor odstránený súborovým systémom. Druhý spôsob predstavuje iniciovanie operácie nastavenia informácií (IRP štruktúra s označením IRP_MJ_SET_INFORMATION) so špecifikovaným príznakom dispozície určujúcej odstránenie súboru.

Jadro pieskoviska registruje operáciu nastavenia informácií a selektuje štruktúru dispozície. Následne je celá cesta k súboru uložená do tabuľky odstránených súborov. Tabuľka obsahuje cestu vždy v tvare hostiteľského umiestnenia. Ak je súbor z pieskoviska, ďalšie vykonávanie je prenechané I/O manažérovi. Ak je súbor z hostiteľského priestoru, jadro iniciuje vlastnú operáciu nastavenia informácií s negatívnym príznakom dispozície určujúcej odstránenie súboru. V závislosti od výsledku jadrom iniciovanej operácie, aplikuje výsledný stav a dokončí pôvodnú operáciu nastavenia informácií.

Jadro spravuje nezávislé pre každé logické pieskovisko tabuľku uchovávajúcu zoznam odstránených súborov, implementačne riešenú ako AVL strom. Synchronizácia prístupu k tabuľke je zabezpečená použitím rýchlych mutexov. Aby bolo možné uchovať stav pieskoviska aj po vypnutí a opätovnom štarte operačného systému, obsah tabuľky je uložený do súboru a počas vytvárania logického pieskoviska opätovne načítaný do pamäte.

Implementácia tabuľky ovplyvnila proces rozhodovania pri presmerovaní popísaný v predchádzajúcej časti. V prípade, že sa súbor otvára je prvým krokom kontrola existencie cesty k súboru v tabuľke. Ak sa súbor nachádza v tabuľke, požiadavka sa zamietne a dokončí operácia. V opačnom prípade postupuje popísaným spôsobom. Ak sa súbor bude s určitosťou vytvárať, kontrola existencie cesty v tabuľke nie je v procese rozhodovania relevantná a v prípade, že sa tam nachádza bude z tabuľky jednoducho odstránená.

4.2.4 Premenované súbory a pevné odkazy

Premenovanie alebo presúvanie patrí medzi štandardne vykonávané operácie na súboroch. Menej zvyčajnou operáciou je vytváranie pevných odkazov (z angl. hard link), ktoré v systémoch Windows podporuje iba súborový systém NTFS. Pevný odkaz reprezentuje stav vo forme súboru, kedy viac ako jedna cesta smeruje k jednému, tomu istému súboru. Pieskovisko tak musí okrem iného kontrolovať premenovania a vytváranie pevných odkazov iniciované izolovanými procesmi. Jadro prednostne zaujímajú prvé tri fázy v procese týchto operácií. Otvorenie súboru, ktorý má byť premenovaný, otvorenie cieľového súboru s novým názvom a vytvorenie IRP paketu predstavujúceho operáciu nastavenia informácií asociovaného so súborovým objektom prvej operácie a obsahujúc nové meno súboru a ďalšie detaily. Prvá fáza v rámci jadra pieskoviska sa vykoná presne podľa vývojového diagramu na Obr. 12. Druhá fáza je mierne odlišná od štandardného otvárania, pretože cieľový súbor neexistuje. V skutočnosti požiadavka obsahuje špeciálny príznak indikujúci vlastnosť, že volajúci proces sa pokúša otvoriť cieľ premenovania alebo vykonať operáciu vytvárania pevných odkazov. Jadro pieskoviska takýto príznak zachytí a bez ďalšej analýzy požiadavky, presmeruje cieľ do domény pieskoviska. Tretia fáza je odlišná v závislosti od toho či ide o operáciu premenovania alebo vytvárania pevných odkazov.

Pri operácii premenovania, v tretej fáze navrhovaný filter získa informácie o zdrojovom súbore. Ak je zdrojový súbor z domény pieskoviska, jadro si môže byť isté, že k modifikácii hostiteľa za žiadnych okolností nedôjde a zvyšnú réžiu prenechá I/O manažérovi. Ak sa zdrojový súbor nachádza u hostiteľa, napr. v prípade, keď v prvej fáze premenovania nebol nastavený príznak na zápis (štandardná situácia pri volaní funkcie API presúvania súboru), jadro získa informácie o cieľovom súbore, skopíruje zdrojový súbor z hostiteľa do domény pieskoviska pod novým menom, pridá zdrojový súbor do tabuľky vymazaných položiek, stornuje ďalšie

vykonávanie a indikuje úspešný stav. Premenovania v rámci diskových zväzkov už z princípu nezávislosti súborových systémov nie sú možné a interne sa vykonávajú ako operácia kopírovania a odstraňovania súborov, čo v konečnom dôsledku značí elimináciu tretej fázy. Premenovania medzi zväzkami tak jadro pieskoviska podporuje implicitne, pretože je dekomponovaná na operáciu kopírovania a odstraňovania.

Vytváranie pevných odkazov je možné iba v rámci jedného zväzku. Ak zdrojový súbor leží na zväzku C, nový pevný odkaz sa môže nachádzať iba na tomto zväzku. Tento fakt ovplyvnil proces spracovania tretej fázy, kedy na rozdiel od operácie premenovania musí jadro pieskoviska kontrolovať zdrojový a cieľový zväzok operácie. Následne ak obe umiestnenia korešpondujú s jedným zväzkom, dochádza k analýze zdrojového súboru. Ak zdrojový súbor je z domény pieskoviska, ďalšie vykonávania je ponechané I/O manažérovi. V opačnom prípade dochádza k vytvoreniu kópie zdrojového súboru do pieskoviska a k systémovému volaniu umožňujúcemu vytvoriť pevný odkaz nie z originálneho, ale z tieňového – skopírovaného súboru. Posledný krok je identický ako u operácie premenovania, kedy dochádza k stornovaniu ďalšieho vykonávania a indikovaniu úspešného výsledného stavu.

4.2.5 Enumerácia adresárov

Enumerácia adresára je jednou z operácií pre získavanie informácií o adresári. Výsledkom operácie je množina súborov a adresárov nachádzajúcich sa v dopytovanom adresári. Pretože navrhované pieskovisko využíva techniku kopírovania súborov pri zápise, ako bolo spomenuté v časti 2.1, súbory sa môžu nachádzať na dvoch umiestneniach. Ak izolovaný proces požiada o informácie o položkách nachádzajúcich sa v konkrétnom adresári, odpoveď musí obsahovať *zjednotenie* oboch umiestnení s vynechaním *duplicitných záznamov* a *záznamov nachádzajúcich sa v tabuľke odstránených súborov*. Duplicitné záznamy predstavujú také súbory a adresáre, ktoré sa nachádzajú v doméne pieskoviska aj v doméne hostiteľa. Odpoveď pri týchto situáciách nebude obsahovať záznamy z domény hostiteľa.

Program žiada o položky adresára prostredníctvom jedného alebo viacerých volaní (funkcia API s názvom NtQueryDirectoryFile). Počet volaní závisí od počtu súborov v adresári, špecifikovanej súborovej maske a veľkosti alokovanej výstupnej štruktúry. I/O manažér pri tomto volaní iniciuje IRP požiadavku týkajúcu sa správy adresárových operácií s označením IRP_MJ_DIRECTORY_CONTROL. Tento IRP paket ďalej obsahuje dve pod operácie. Ak pod operácia tvorí dopyt na záznamy v adresári, systém

rozlišuje deväť štruktúr, z ktorých volajúci program špecifikuje konkrétnu štruktúru vhodnú pre danú situáciu. Podľa veľkosti výstupnej vyrovnávajúcej pamäti, súborovej maske, a počte položiek, súborový systém nakopíruje informácie vo forme jednej z deviatich štruktúr do výstupnej pamäti, pričom každá obsahuje okrem informácií o položke v adresári aj posun určujúci nasledujúci záznam. Jednotlivé volania funkcie sú stavové. To znamená, že každé ďalšie volanie funkcie vráti nasledujúce položky pokiaľ volajúci proces explicitne nešpecifikuje inak alebo volanie vráti stav indikujúci situáciu, že už pre dané parametre nie sú žiadne ďalšie súbory.

Enumerácia adresárov je implementačne najzložitejšia časť filtrovania súborov. Jadro pieskoviska registruje *post-operáciu správy adresárových operácií* a filtruje šesť štruktúr súvisiacich s menami súborov. Algoritmus rozlišuje dva základné prípady, ktoré môžu pri enumerácií nastať. V situácií, že súborový objekt reprezentuje adresár z hostiteľského umiestnenia je zrejmé, že korešpondujúci adresár v pieskovisku neexistuje, pretože nebolo vykonané presmerovanie. Následne sa pre každú položku kontroluje výskyt v tabuľke odstránených súborov. Ak záznam existuje v tabuľke, všetky položky vo výstupnej vyrovnávajúcej pamäti sa presunú pred odstránenú položku pomocou špecifikovaného posunu. Komplikovanejšie spracovanie nastáva ak súborový objekt reprezentuje adresár z domény pieskoviska. Ak volanie nastalo prvýkrát, otvorí sa a získa súborový deskriptor k adresáru z ekvivalentného hostiteľského umiestnenia. Deskriptor sa pre použitie v ďalších volaniach uloží do SHC kontextu. Všetky položky z pieskoviska vrátené súborovým systémom sa vložia do pomocnej tabuľky. Tabuľka uchováva iba mená súborov a vnútorne je implementovaná ako spájaný zoznam uchovávajúci si referenciu na prvý a posledný prvok v zozname. Po vrátení všetkých položiek súborový systém oznámi, že už nie sú žiadne ďalšie súbory v adresári. V ďalšom kroku jadro začne iniciovať vlastné požiadavky pre získanie položiek nachádzajúcich sa v hostiteľskom umiestnení. Parametre systémového volania vrátane súborovej masky a veľkosti výstupnej vyrovnávajúcej pamäte sú identické s parametrami pôvodného volania. Ak filtrom iniciované volanie vráti úspešný stav, výstupná vyrovnávajúca pamäť obsahuje záznamy z hostiteľského umiestnenia. Pre každý záznam sa skontroluje existencia v pomocnej tabuľke duplicít a v tabuľke odstránených súborov. Ak je výskyt negatívny, záznam sa skopíruje do výstupnej vyrovnávajúcej pamäti pôvodného volania. Pretože pôvodné volanie neindikovalo žiadne ďalšie položky, výstupná vyrovnávacia pamäť je prázdna. Po overení všetkých položiek, filter zmení výsledný stav pôvodného volania na úspešný stav. Popísaný

algoritmus pokračuje dovtedy, pokiaľ nebudú vrátené všetky položky z hostiteľského umiestnenia alebo proces preruší enumeráciu.

4.3 Filtrovanie registrov

Registre systému Windows obsahujú kritické časti pre spúšťanie operačného systému a jeho nastavenia. Ošetrenie prístupu k registrom je preto dôležitou súčasťou navrhovaného riešenia. Podobne ako pri virtualizácii súborového prístupu, filtrovanie registrov je založené na technike kopírovaní kľúčov pri zápise resp. v prípade registrov pri zmene hodnôt. Na rozdiel od súborov je pri registroch implementovaný tento mechanizmus v plnom zmysle slova. Kľúč a hodnota sú v pieskovisku vytvorené až v prípade, kedy dochádza k vytvoreniu kľúčov a hodnôt resp. ich zmene.

Pieskovisko pre registre reprezentuje samostatná vetva umiestnená v kľúči `HKEY_LOCAL_MACHINE\Software\ESU`. V režime používateľa, systém Windows rozlišuje sedem predefinovaných koreňových kľúčov, z ktorých päť sa štandardne zobrazuje v editore registrov: `HKEY_LOCAL_MACHINE` (HKLM), `HKEY_USERS` (HKU), `HKEY_CLASSES_ROOT` (HKCR), `HKEY_CURRENT_USER` (HKCU) a `HKEY_CURRENT_CONFIG` (HKCC). V skutočnosti registre obsahujú iba prvé dva korene. Ostatné koreňové kľúče sú symbolické odkazy. Zobrazenie ciest medzi originálnym koreňovým kľúčom a ekvivalentným kľúčom v pieskovisku s označením „pieskovisko“ sa nachádza v Tab. 2.

Tab. 2 Originálny koreňový kľúč a jeho zobrazenie v pieskovisku

Koreňový kľúč	**Ekvivalentný kľúč v pieskovisku**
HKLM	HKLM\Software\ESU\pieskovisko\Registry\Machine
HKU	HKLM\Software\ESU\pieskovisko\Registry\User
HKCR	HKLM\Software\ESU\pieskovisko\Registry\Machine\Software\Classes
HKCU	HKLM\Software\ESU\pieskovisko\Registry\User*{SID aktuálneho používateľa}*
HKCC	HKLM\Software\ESU\pieskovisko\Registry\Machine\System\currentcontrolset\hardware profiles\current

4.3.1 Základná architektúra filtrovania registrov

Filter registrov je akýkoľvek ovládač, ktorý filtruje volania registrov. Konfiguračný manažér (CM), ktorý implementuje registre, dovoľuje filtrovať akékoľvek volania procesu týkajúcich sa registrov. Filtrovanie registrov v základnej podobe je možné od verzie Windows XP. Použiteľné rozhranie a množstvo vylepšení pre modifikáciu a odchytávanie volaní priniesol až systém Windows Vista.

Podobne ako pri súboroch, registrový filter prijíma pre-notifikácie a post-notifikácie. Ovládač zachytáva *všetky* notifikácie. Ak konkrétna notifikácia nie je pre ovládač relevantná, prinajmenšom musí odovzdať vykonávanie naspäť manažérovi konfigurácie. Ak operáciu filtruje počas pre-notifikácie, môže vykonať požadované spracovanie a vrátiť jeden z troch stavov:

- *úspešný stav*, kedy registrovú operáciu spracuje CM a následne iniciuje post-notifikáciu obsahujúcu požadované údaje a výsledný stav operácie,
- *neúspešný stav*, kedy dôjde k zamietnutiu operácie a okamžitému vráteniu chybového stavu procesu bez volania post-notifikácie a asistencie CM,
- *obchádzajúci stav*, kedy všetko spracovanie musí vykonať ovládač pričom registrová operácia sa skončí úspešne a bez iniciovania post-notifikácie a asistencie CM.

Filtre registrov sú usporiadané v zásobníku podľa nadmorskej výšky. Ovládač si môže zaregistrovať niekoľko registrových filtrov a monitorovať, blokovať alebo modifikovať údaje. Podobne ako pri minifiltroch aj pri filtroch registrov existujú kontexty. Aktuálne je možné kontext asociovať s registrovým filtrom, registrovou operáciou (notifikáciou) a registrovým objektom reprezentujúci otvorený kľúč.

Navrhované pieskovisko filtruje niekoľko základných notifikácií súvisiacich s otváraním, vytváraním, odstraňovaním, dopytovaním a enumeráciou kľúčov a hodnôt. Počas filtrovania registrov treba brať na zreteľ množstvo faktorov a možných problémov. Prístup k vyrovnávajúcej pamäti alokovanej procesom pre danú operáciu nie je vždy bezpečný a záleží od verzie OS. Modifikácia údajov a samostatné spracovanie operácie ovládačom bez účasti CM musí ošetriť a vrátiť správne hodnoty vrátane bezpečnostnej politiky. Dodatočné spracovanie registrov musí byť čo najrýchlejšie. Windows intenzívne využíva prístup k registrom. Príkladom môže byť 20 000 filtrovaných registrových prístupov počas otvárania a výberu jedného súboru

v poznámkovom bloku. Z uvedených dôvodov je filtrovanie registrov komplexnou a rozsiahlou časťou ovládača.

4.3.2 Otváranie a vytváranie kľúčov

Rozhranie pre filtrovanie registrov na rozdiel od súborov rozlišuje otváranie a vytváranie kľúčov. Pri vytváraní kľúčov ak kľúč neexistuje, je vytvorený a následne otvorený. V prípade ak proces otvára kľúč, jadro vytvorí dve cesty. Cestu reprezentujúcu prístup k originálnemu kľúču a cestu ku kľúču v pieskovisku. Ak kľúč v pieskovisku neexistuje a aktuálny kľúč sa otvára absolútne, ďalšia réžia je prenechaná na CM. V prípade ak sa operácia skončila úspešne, v post-notifikácii sa nastaví kontext pre nový registrový objekt a uložia sa obe získané cesty. Ak kľúč v doméne pieskoviska existuje, jadro ho otvorí, vyplní požadované údaje, nastaví kontext pre registrový objekt reprezentujúci kľúč v pieskovisku a vráti obchádzajúci stav. Operácia vytvorenia kľúča je implementovaná podobne. Ak kľúč v pieskovisku existuje, vykonávanie je identické s operáciou otvorenia. V opačnom prípade sa overuje existencia kľúča u hostiteľa. Pri pozitívnom výskyte je ďalšie vykonávanie rovnaké ako pri otváraní. Inak jadro vytvorí kľúč v pieskovisku, nastaví požadované údaje, asociuje kontext pre registrový objekt reprezentujúci kľúč v pieskovisku a vráti obchádzajúci stav. Vo všetkých prípadoch je zahrnutá podpora tabuľky uchovávajúcich odstránené kľúče a hodnoty. To znamená kontrolu tabuľky počas otvárania kľúča a odstránenie záznamu z tabuľky pri vytvorení kľúča.

4.3.3 Zmena hodnôt

Hodnoty sa v kontraste s kľúčmi neotvárajú. Hodnota môže byť vytvorená a čítaná iba v rámci otvoreného kľúča. Ak sa hodnota nachádza pod kľúčom v pieskovisku, ďalšie filtrovanie nie je potrebné. Konfiguračný manažér nastaví novú hodnotu a jadro v prípade výskytu hodnoty v tabuľke odstránených kľúčov a hodnôt ,hodnotu z tabuľky odstráni. Ak sa hodnota nachádza v hostiteľskom kľúči, uplatňuje sa všeobecný princíp kopírovania pri zápise. V pieskovisku sa vytvoria všetky ekvivalentné kľúče na ceste k hodnote. Následne sa vytvorí aj hodnota s parametrami originálneho volania. Posledným krokom je odstránenie záznamu z tabuľky pri pozitívnom výskyte hodnoty.

4.3.4 Odstraňovanie kľúčov a hodnôt

Základný princíp správania pri odstraňovaní kľúčov a hodnôt je rovnaký ako u súborov. Jadro si uchová zoznam vymazaných kľúčov a hodnôt v tabuľke popísanej

v časti 4.2.3. Pri notifikácií o odstránenie kľúča jadro rozlišuje pôvod kľúča. Ak je kľúč z pieskoviska, ďalšie vykonávanie je prenechané na CM. Ak kľúč pochádza z domény hostiteľa, jadro vráti obchádzajúci stav. V oboch prípadoch je do tabuľky vložená plná cesta ku kľúču.

Odstraňovanie hodnôt závisí od kľúča, v ktorom sa nachádzajú. Ak je kľúč z domény hostiteľa a hodnota existuje, jadro zabráni odstráneniu hodnoty a uloží cestu k hodnote do tabuľky. Ak je kľúč z pieskoviska a hodnota existuje, vykonávanie je prenechané na CM a záznam sa uloží do tabuľky. Ak hodnota v pieskovisku neexistuje, jadro skontroluje výskyt hodnoty pod kľúčom hostiteľa. Pri pozitívnom výskyte jadro zamedzí odstráneniu hodnoty z hostiteľa, uloží záznam do tabuľky a notifikuje proces o úspešnej operácii. V opačnom prípade je bezpečné ďalšie vykonávanie nechať na konfiguračnom manažérovi. Keďže kľúče a hodnoty môžu mať rovnaký názov v rámci totožnej cesty, v tabuľke je potrebné rozlišovať medzi kľúčmi a hodnotami. Všetky hodnoty v tabuľke majú na konci pridané dve spätné lomky, napr. hodnota s menom „rok“ nachádzajúca sa v kľúči sa názvom „Svet“ v koreňovom kľúči HKLM a kľúči „Software“ je v tabuľke uložená vo formáte

```
\Registry\Machine\Software\Svet\rok\\
```

pričom kľúč s rovnakým názvom „rok“ má v tabuľke tvar

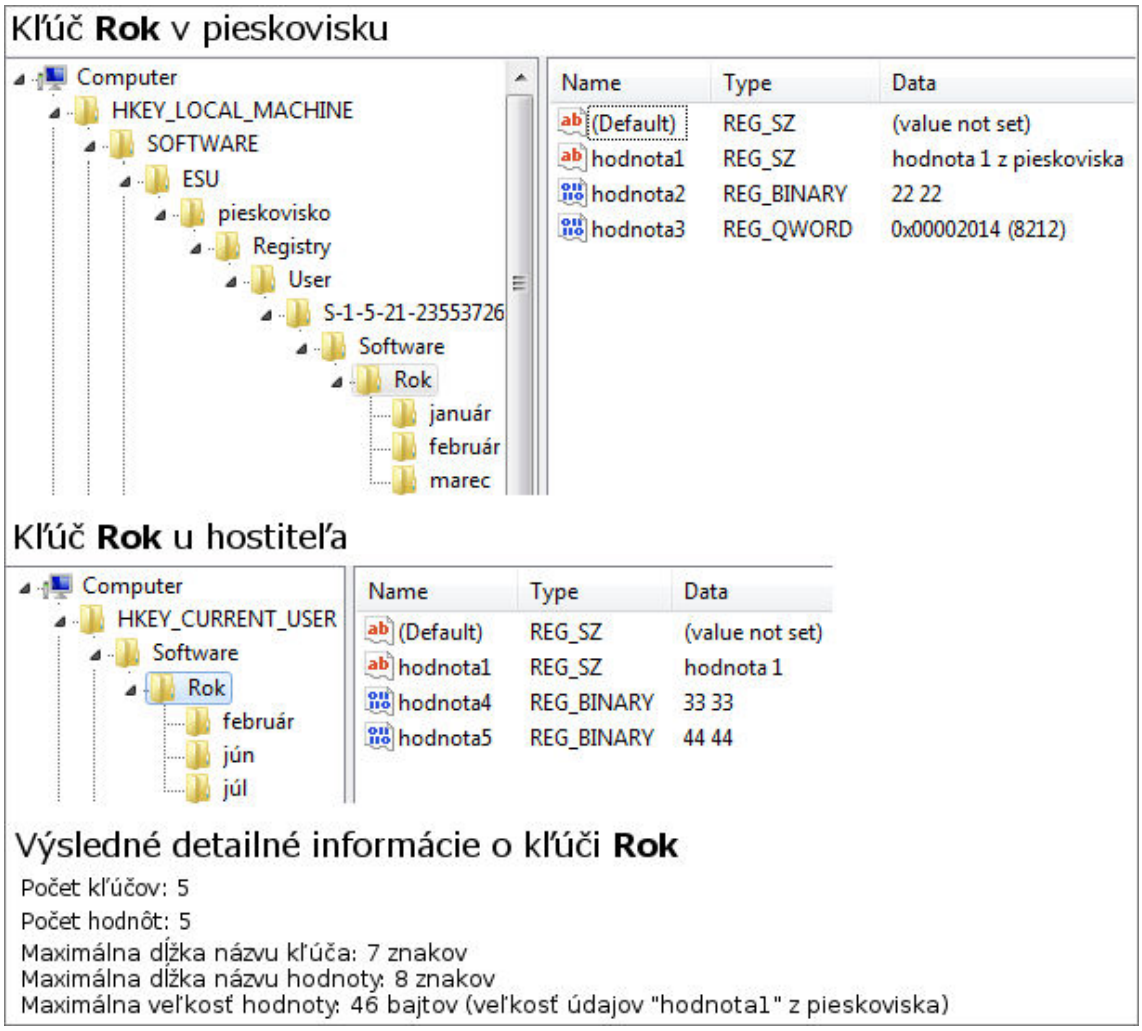
```
\Registry\Machine\Software\Svet\rok
```

4.3.5 Požiadavky na informácie o kľúčoch a hodnotách

Pri získavaní informácií o kľúčoch a hodnotách rozlišujeme štyri základné registrové operácie: dopyty na kľúče, dopyty na hodnoty, enumerácia kľúčov a enumerácia hodnôt. Jadro filtruje všetky operácie.

Pri dopytoch na kľúče, dopytujúci proces špecifikuje jednu zo šiestich štruktúr podľa toho, aký druh informácií o kľúči potrebuje. Ak požaduje získanie názvu kľúča a kľúč sa nachádza v pieskovisku, jadro musí zmeniť názov kľúča, ktorý ma formát plnej cesty v rámci pieskoviska na hostiteľský názov. Druhým typom údajov sú detailné informácie o kľúči, ktoré ukrývajú údaje o počte podkľúčov a hodnôt, o maximálnej dĺžke názvu podkľúča a hodnoty a podobne. Pretože sa kľúče môžu nachádzať na dvoch miestach, jadro dopytuje hostiteľský a ekvivalentný kľúč v pieskovisku a pre každý podkľúč z hostiteľa overuje výskyt v tabuľke odstránených kľúčov a taktiež či podkľúč nie je duplikát s podkľúčom z pieskoviska. Identický postup je aplikovaný pri počítaní hodnôt. Po spočítaní podkľúčov a hodnôt, určení maximálnych dĺžok názvov

elementov, jadro skopíruje údaje do výstupnej štruktúry a vráti obchádzajúci stav. Príklad dopytu pre získanie detailných informácií o kľúči „Rok“ nachádzajúceho sa v pieskovisku a u hostiteľa ilustruje Obr. 13.



Obr. 13 Príklad dopytu pre získanie detailných informácií o kľúči

Dopytovanie hodnôt je na rozdiel od kľúčov iba čítanie ich dát. Pre získavanú hodnotu súvisiacu s kľúčom hostiteľa sa jednoducho overuje existencia hodnoty v tabuľke odstránených hodnôt. Pri hodnote dopytovanej v rámci kľúča z pieskoviska okrem kontroly tabuľky sa v prípade neexistencie hodnoty prehliada hodnota z kľúča v hostiteľovi. Ak hodnota existuje, údaje sa skopírujú do výstupnej pamäte. V opačnom prípade ďalšie spracovanie vyrieši CM.

Enumerácia kľúčov a hodnôt podobne ako enumerácia u súborov predstavuje najzložitejšiu časť filtrovania registrov. Požadované správanie je rovnaké ako pri súboroch – zjednotenie podkľúčov resp. hodnôt s vylúčením duplicitných záznamov a položiek nachádzajúcich sa v tabuľke odstránených kľúčov a hodnôt. Enumerácia

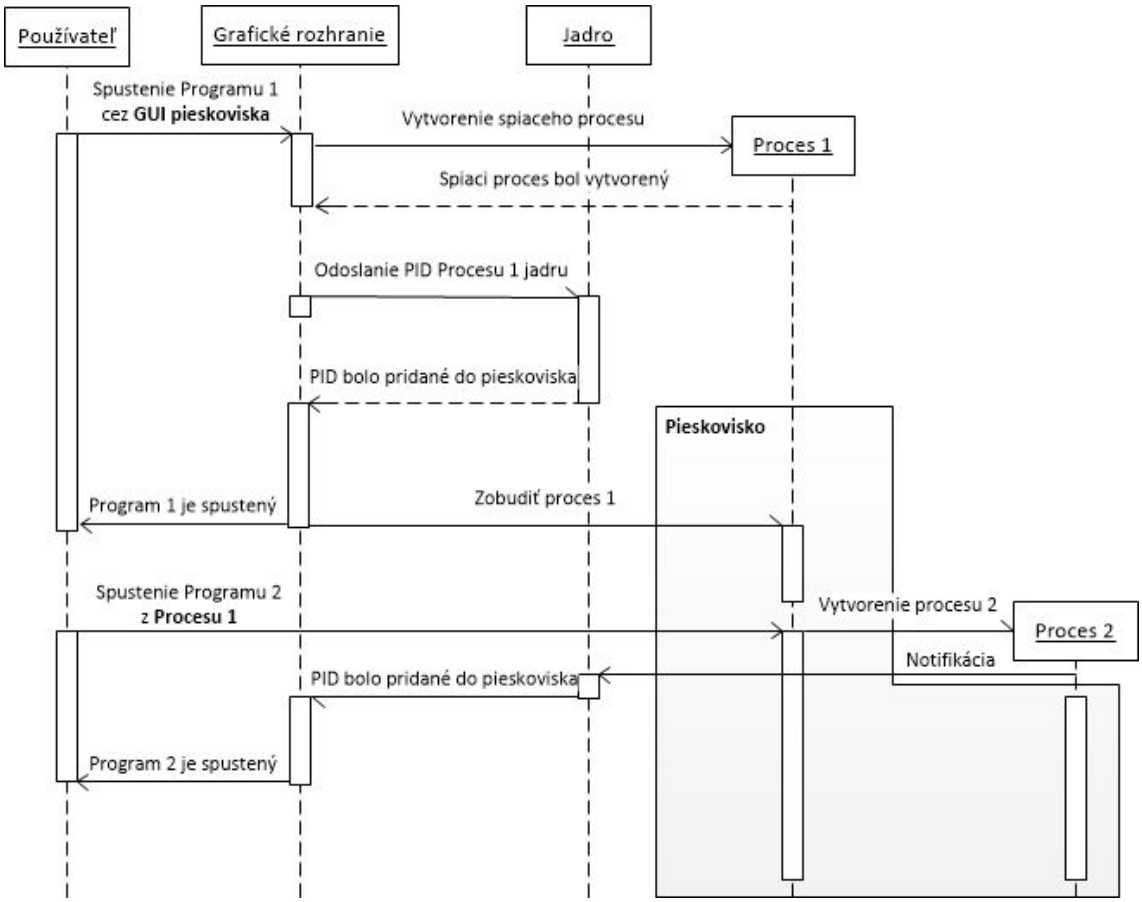
kľúčov v operačnom systéme Windows funguje odlišným spôsobom ako enumerácia súborov. Pri jednom volaní funkcie pre enumeráciu kľúča, volanie vráti *maximálne jeden záznam*. Okrem toho, volanie *nie je stavové*, t.j. opakované volanie funkcie nevracia nasledujúci kľúč. Pri každom volaní je tak potrebné špecifikovať index podkľúča. Spomenuté faktory ovplyvnili implementáciu filtrovania enumerácie. Ak kľúč, ktorého obsah sa získava pochádza z domény pieskoviska, jadro dopytuje podkľúč s rovnakými vstupnými parametrami a ukladá získané meno kľúča do pomocnej tabuľky pre hľadanie duplikátov. Každé ďalšie volanie prebieha rovnako až pokiaľ kľúč z pieskoviska nevypíše všetky podkľúče. Následne jadro získava podkľúč z kľúča u hostiteľa. Overí sa či je to duplikát alebo sa nachádza v tabuľke odstránených kľúčov. Pokiaľ sa nenájde podkľúč, ktorý sa nenachádza v oboch tabuľkách, iterácia podkľúčov pokračuje. Vhodný záznam sa skopíruje do výstupnej pamäte a jadro vráti obchádzajúci stav. Okrem toho, do kontextu asociovanom s registrovým objektom reprezentujúci kľúč, sa uloží *posun indexu* inkrementovaný indexom podkľúča z hostiteľského kľúča, a počtom duplicitných resp. vymazaných kľúčov. Pri ďalšom volaní funkcie enumerácie, jadro k vstupnému indexu pripočíta posun a vráti správny záznam. V prípade, že kľúč na ktorom je vykonávaná enumerácia pochádza z hostiteľa, kontroluje sa iba tabuľka odstránených kľúčov a podľa potreby aktualizuje posun indexu. Enumerácia hodnôt je implementovaná rovnakým spôsobom, avšak sú získavané hodnoty, nie kľúče.

4.4 Správa pieskovísk a procesov

Návrh pieskoviska od počiatku rátal s podporou viacerých pieskovísk. To znamená množstvo vzájomných izolovaných prostredí koexistujúcich v rámci jedného OS. V ovládači sa nachádza globálna tabuľka uchovajúca informácie o jednotlivých pieskoviskách. Každé pieskovisko je jednoznačne identifikované *číselným identifikátorom*. Ďalej obsahuje meno pieskoviska, cestu určujúcu koreňový adresár na pevnom disku a koreňový kľúč k registrom a celkový počet procesov nachádzajúcich sa v danom pieskovisku. Takmer všetky funkcie v jadre pracujú s konkrétnym pieskoviskom pomocou identifikátorov. Ovládač obsahuje metódy pre vytváranie, odstraňovanie a získavanie logických pieskovísk.

Ovládač, minifilter, registrový filter alebo iný komponent štandardne zachytáva všetky volania v systéme. Cieľom riešenia je filtrácia konkrétnych programov. Jadro pieskoviska musí determinovať kontext aktuálne vykonávaného vlákna pre všetky

spätné volania a asociovať ho s procesom. Proces v operačnom systéme je jednoznačne určený číselným identifikátorom PID. Windows umožňuje pre každý ovládač zaregistrovať tzv. spätné volanie iniciované pri vytváraní a ukončovaní procesov. Klientska časť pri spustení procesu odošle PID jadru, ktoré si ho uloží do globálnej tabuľky procesov. Spätné volanie obsahuje okrem identifikátora aktuálne vytváraného procesu aj PID rodiča, ktorý proces vytvoril. Ak izolovaný proces vytvorí ďalší proces, jadro pieskoviska pomocou hodnoty rodičovského PID spätného volania identifikuje nový proces a pridá ho do tabuľky procesov. Mechanizmus je znázornený sekvenčným diagramom na Obr. 14.



Obr. 14 Spustenie procesov v pieskovisku

Prístup k tabuľke procesov musí byť čo najrýchlejší a odolný voči chybám. Štruktúra je implementovaná ako statická hašovacia tabuľka. Synchronizácia je vyriešená prostredníctvom mechanizmu e-zdrojov (z angl. EResource), ktoré umožňujú súbežné čítanie a exkluzívny zápis do tabuľky. V prípade ak dôjde k odstráneniu pieskoviska a pieskovisko obsahuje spustené procesy, jadro ich okamžite terminuje.

5 Klientska časť

Obsiahnutie všetkých aspektov potrebných pre vývoj bezpečnostného systému na báze modifikácií alebo filtrovaní systémových volaní nie je možné dosiahnuť iba prostredníctvom minifiltrov. V nadväznosti na časť 3.3 je zrejmé, že obmedzenia systémových zdrojov, sieťová izolácia a ďalšie nekritické prerušenia sú implementované v režime používateľa. Uvedené technológie štandardne vyžadujú minimálne administrátorské oprávnenia alebo vykonávanie v kontexte systémového procesu. Pre splnenie týchto požiadaviek bola navrhnutá služba systému Windows.

Ovládače v operačnom systéme Windows neposkytujú rozhranie pre komunikáciu s používateľom. Grafický subsystém Win32 a subsystém Windows POSIX sú implementované v režime používateľa, nakoľko ovládače vykonávané v režime jadra nemajú k týmto subsystémom prístup. Aj keď systém môže zaviesť ovládač počas štartu systému, v prípade navrhovaného riešenia, bez ďalších inštrukcií jadro pieskoviska nemôže filtrovať systémové volania špecifického procesu. Okrem toho, kvôli podpore viacerých logických pieskovísk a možnosti prispôsobenia, bola navrhnutá používateľská aplikácia ovládajúca minifilter a komunikujúca s používateľom. Používateľská aplikácia by sa nemala z bezpečnostných dôvodov vykonávať pod právami administrátora.

Moderné a atraktívne používateľské rozhranie implementované v rámci štandardného Windows API v jazyku C/C++ je komplikovaný a časovo náročný proces. V súčasnosti sa intenzívne využívajú technológie .NET. Služba systému Windows aj používateľské rozhranie sú implementované prostredníctvom týchto technológií.

5.1 Komunikácia jadra s používateľskou aplikáciou

Jadro pieskoviska komunikuje s používateľskou aplikáciu pomocou rámca pre komunikáciu minifiltrov. Manažér filtrov obsahuje rozhranie, ktoré abstrahuje komunikáciu na použitie synchrónnych a asynchrónnych správ. Používateľská aplikácia komunikuje s jadrom pieskoviska väčšinou synchrónne, kedy očakáva odpoveď. Minifiltre si odovzdávajú údaje najčastejšie pomocou klasických štruktúr jazyka C. Jadro pieskoviska aktuálne rozlišuje osem kontrolných kódov a pozná deklaráciu zodpovedajúcich štruktúr týkajúcich sa: vytvárania, odstraňovania a premenovania pieskovísk, nastavenia a vytvorenia potrebných adresárov, pridávania procesov do pieskovísk, získavania informácií o diskových oddieloch a nakoniec jednoduché ovládanie zaznamenávania činnosti procesov. Manažované aplikácie, t.j. aplikácie

vyžadujúce pre svoje vykonávanie virtuálny stroj resp. aplikácie napísane v jazyku C#, Visual Basic a podobne, nemôžu priamo využívať natívny kód. Jednou z možností je použitie techniky PInvoke, kedy sa importujú funkcie z natívnych knižníc do manažovaného jazyka. V navrhnutom riešení je použité elegantnejšie riešenie. Medzivrstva medzi používateľským rozhraním a minifiltrom tvorí *samostatná knižnica napísaná v jazyku C++/CLI*. V tomto jazyku je možné využívať štruktúry a konštrukcie natívnych jazykov (C resp. C++) a taktiež manažované prostredie poskytnuté virtuálnym strojom. Navrhnutá knižnica implicitne konvertuje štruktúry jednotlivých kontrolných kódov na manažované inštancie triedy a naopak. Okrem toho, medzivrstva slúži pre pripojenie a udržanie spojenia medzi GUI a jadrom. Jadro pieskoviska z bezpečnostných dôvodov obmedzuje počet pripojených aplikačných procesov na jednu inštanciu.

5.2 Windows služba

Prvý krok determinujúci možnosť použiť grafickú aplikáciu je načítanie ovládača. Používateľská aplikácia štandardne nemá dostatočné práva pre načítanie ovládača do systému. Ak pri štarte GUI zlyhá snaha o pripojenie na ovládač, GUI odošle správu službe s požiadavkou na načítanie minifiltra. Komunikácia sa uskutočňuje pomocou technológie WCF (z angl. Windows Communication Foundation) dostupnej od verzie .NET 3.0. WCF je aplikačné programové rozhranie pre vytváranie prepojených aplikácií orientovaných na služby (v inom zmysle slova ako Windows služby). Klientska časť pieskoviska využíva možnosť WCF abstrahovať komunikáciu cez pomenované rúry na jednoduchú výmenu manažovaných objektov prostredníctvom spoločného, popísaného rozhrania.

Služba pieskoviska obsahuje plnú podporu pre používateľské prerušenia. Pomocou rozhrania knižnice EasyHook je implementovaná DLL knižnica s návestiami na prerušené funkcie, ktorá je v prípade potreby vložená do adresného priestoru procesu spusteného v pieskovisku. Aktuálne je implementované iba prerušenie funkcie pre nastavenie titulného textu hlavného okna izolovaného procesu. Ďalšie prerušenia, napr. ošetrenia komunikácie procesov pomocou Windows správ je možné implementovať v budúcnosti pomocou vopred pripravenej DLL knižnice.

5.3 Správa zdrojov

Obmedzenia systémových zdrojov a práv sú v niektorých prípadoch dôležitou súčasťou ochrany. V pieskovisku je implementovaná podpora pre kontrolu zdrojov pomocou Windows Job objektov, ktoré tvoria súbor procesov. Táto technológia umožňuje prideliť jednotlivé procesy do kontajnerov a aplikovať požadované vlastnosti [35]. Objekt Job-u dovoľuje spravovať procesy ako uniformnú jednotku. Množina obmedzení môže byť nastavená v rámci jedného objektu, ktorý donúti použiť obmedzenie pre každý proces v súbore procesov.

Pieskovisko umožňuje používateľovi špecifikovať politiku systémových zdrojov pre každé logické pieskovisko. Súčasné obmedzenia systémových zdrojov pre pieskovisko zahrňujú: maximálne využitie operačnej pamäte, zmena priority procesov, maximálne vyťaženie procesora a maximálny počet súbežných procesov. Okrem obmedzení systémových zdrojov, pieskovisko umožňuje zakázať tieto operácie: zmena systémových nastavení, vypnutie a reštart počítača, čítanie a zapisovanie do schránky systému Windows a aplikovať vlastné používateľské prerušenia. Správa zdrojov je implementovaná v používateľskom rozhraní, pretože procesy štartuje GUI a obmedzenia procesov sú aplikované na procesy okamžite po ich štarte, čo eliminuje potrebnú réžiu v prípade komunikácie so službou.

5.4 Prístup na internet

Vytvorenie kompletného filtra pre prístup na internet by vyžadovalo naprogramovanie zodpovedajúceho ovládača. Využiť by sa mohol model NDIS ovládačov alebo použiť platformu Windows pre filtrovanie sieťovej prevádzky. Tvorba sieťového filtra by bola mimo rozsah diplomovej práce. Pieskovisko využíva služby integrovaného firewallu s rozšíreným rozhraním dostupným od verzie Windows Vista. Windows exportuje programové rozhranie pre ovládanie firewallu. Ovládanie firewallu je implementované v službe pieskoviska pretože vkladanie pravidiel do firewallu vyžaduje oprávnenia administrátora. Služba príjme správu od používateľského rozhrania s názvom spusteného programu. Následne ak neexistuje, vytvorí pravidlo s cestou k aplikácií. Služba si udržiava zoznam pravidiel s aktuálne vykonávanými programami. Ak sa vykonávanie procesu ukončí, služba príjme požadovanú správu a odstráni pravidlo. Uvedený spôsob má viacero nevýhod. Pravidlá nie je možné vytvárať pre konkrétne procesy (ich identifikátory), ale iba pre konkrétne programy t.j.

cesty k spustenej aplikácií. To znamená, že spustenie rovnakého programu v pieskovisku a u hostiteľa spôsobí obmedzenie sieťovej prevádzky pre oba procesy. Taktiež je nevýhodou nemožnosť obmedziť prístup individuálne pre logické pieskoviska, ale iba globálne. Napriek spomenutým nedostatkom je vo väčšine prípadov tento systém pre menej skúsených používateľov dostatočný. Pokročilý používateľ môže túto funkčnosť deaktivovať a použiť vlastné riešenie.

5.5 Používateľské rozhranie

Používateľské rozhranie tvorí bránu medzi používateľom a používaním pieskoviska. V nasledujúcich častiach sú popísané základné časti, princípy a možnosti navrhovaného GUI.

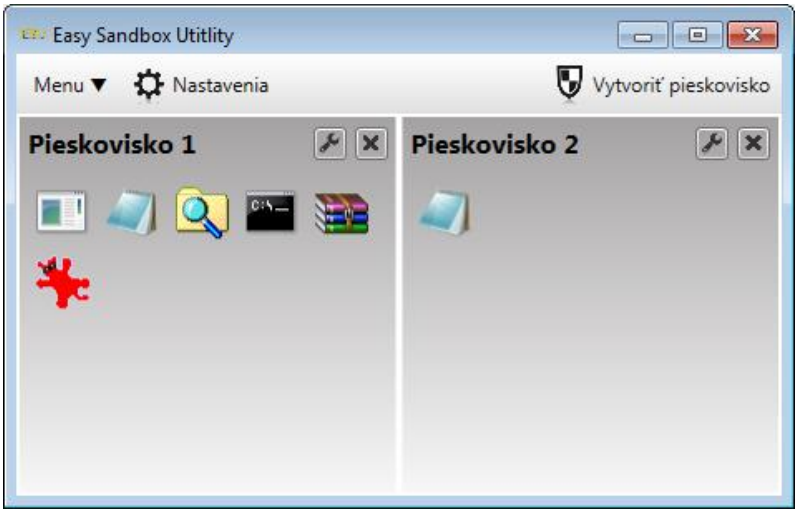
5.5.1 Výber technológie

Hlavný dôraz pri návrhu grafického rozhrania bol kladený na intuitívnosť, pohodlné ovládanie a príjemne prostredie. V súčasnosti pre vývoj systémovej aplikácie s grafickým rozhraním ponúka Microsoft tri základné smery. Použiť tradičné programové rozhranie WinAPI v spojení s jazykom C/C++, použiť objektovú nadstavbu MFC v rámci jazyka C++ alebo využiť služby balíka .NET. Úplne odlišný prístup by znamenal využiť knižnicu tretej strany. Pre rýchly vývoj GUI a funkčnej logiky je v používateľskom rozhraní pieskoviska použitá technológia WPF (z angl. Windows Presentation Foundation) v kombinácií s jazykom C#, štandardne prítomná v .NET od verzie 3. Podľa softvérového architekta a dizajnéra .NET technológií Ch. Andersona [36] predstavuje WPF evolučnú technológiu v používateľskom rozhraní. WPF je prezentačný systém pre tvorbu Windows aplikácií s dôrazom na vizuálne atraktívny a čo najsilnejší používateľský zážitok. Jadrom WPF je vektorový zobrazovací systém akcelerovaný grafický hardvérom. WPF oddeľuje používateľské rozhranie napísané deklaratívne v značkovacom jazyku XAML od aplikačnej logiky naprogramovanej v jazyku C# alebo Visual Basic.

5.5.2 Naviazanie spojenia

Používateľské rozhranie k svojej činnosti potrebuje ovládač. Prvým krokom pri štarte je pokus o naviazanie spojenia. Ak ovládač nie je načítaný, používateľské rozhranie sa pripojí na bežiacu službu a požiada o načítanie ovládača. Vzápätí sa opätovne pokúsi o pripojenie na ovládač. Ak aj druhý pokus zlyhá, pieskovisko informuje používateľa a ukončí svoju činnosť. Používateľské rozhranie nutne

nepotrebuje k vykonávaniu spustenú službu. Bez bežiacej služby, ale nie je možné v prípade potreby načítať ovládač, pridávať programy do firewallu a aplikovať používateľské prerušenia, napr. zmenu titulky okna procesu v pieskovisku. Po úspešnom naviazaní spojenia s ovládačom sa zobrazí hlavné okno aplikácie. Aplikácia so spustenými procesmi a vytvorenými pieskoviskami je zobrazená na Obr. 15.

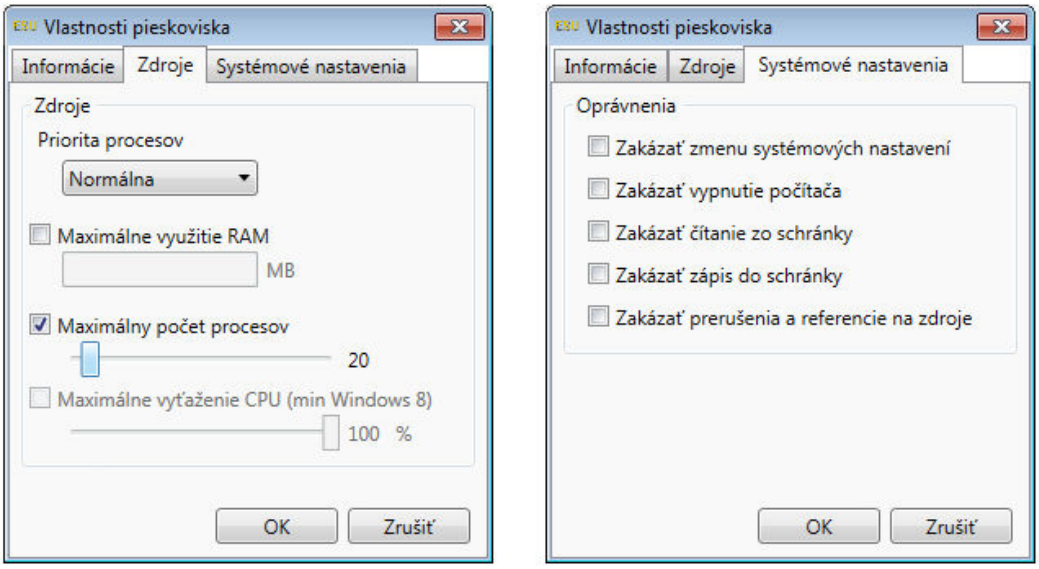


Obr. 15 Používateľské rozhranie pieskoviska so spustenými procesmi

5.5.3 Práca s pieskoviskami

Dizajn používateľskej aplikácie primárne vychádzal z faktu, že jadro pieskoviska podporuje viac ako jedno logické pieskovisko. Používateľ môže logické pieskoviská vytvárať, odstraňovať a konfigurovať. Každé pieskovisko je reprezentované obdĺžnikom a identifikované názvom. Pri vytváraní používateľ špecifikuje požadovaný názov. Následne aplikácia odošle údaje jadru, ktoré ich spracuje. Po úspešnej výmene informácií používateľské rozhranie vykreslí obdĺžnik reprezentujúci izolované prostredie pripravené pre spustenie programov alebo ďalšie nastavenia. Proces vytvárania a odstraňovania pieskovísk je sprevádzaný jednoduchými animáciami, vďaka čomu by mal byť názornejší a prehľadnejší.

Logické pieskovisko môže byť nakonfigurované podľa potrieb používateľa. Základné voľby pre obmedzenia systémových zdrojov sú znázornené na Obr. 16. Aplikácia si aj po ukončení pamätá vytvorené logické pieskoviská a ich nastavenia.

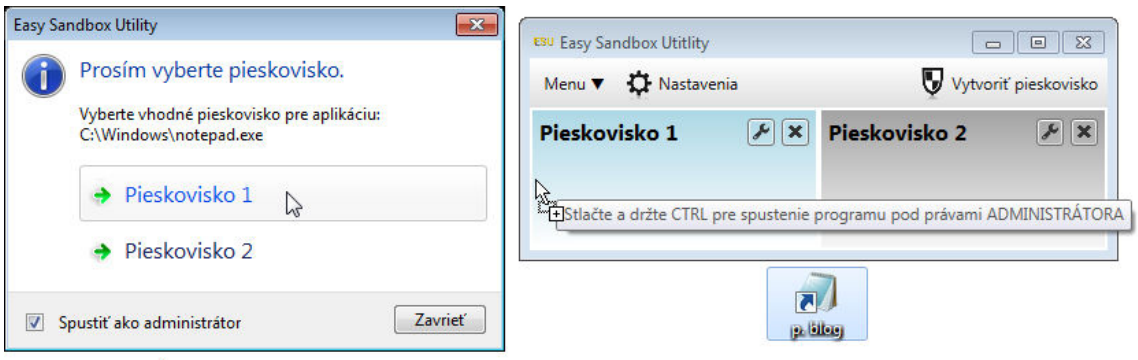


Obr. 16 Konfigurácia logického pieskoviska

5.5.4 Štart programov a ich vizualizácia

Štart programov a vizualizácia ich procesov tvorí dôležitú časť aplikácie. Používateľ musí vedieť, ktoré procesy sú vykonávané kontrolovane a v akom pieskovisku.

Používateľ môže spustiť program dvoma spôsobmi. Prvý spôsob je pomocou štandardného dialógu pre otvorenie súboru. V niektorých prípadoch je rýchlejšia metóda využitím techniky „Drag and Drop“, kedy požadovaný spustiteľný súbor je myšou prenesený nad zvolené pieskovisko. Niektoré programy vyžadujú pre svoje vykonávanie práva administrátora. Používateľské rozhranie pri oboch metódach znázornených na Obr. 17, umožňuje spustiť program so zvýšenými oprávneniami.



a) Štandardný dialóg **b) Drag and Drop**

Obr. 17 Spustenie programu s právami administrátora pomocou dvoch metód

Základnou podmienkou pri spustení aplikácie v pieskovisku je, aby nový proces ihneď po vytvorení neovplyvnil hostiteľský systém. Proces sa musí vytvoriť v spiacom stave. Dôvodom je čas potrebný na doručenie identifikátora PID jadru a službe. Ako

bolo znázornené sekvenčným diagramom na Obr. 14, až po prijatí odpovede o úspešnom pridaní identifikátora do jadra môže byť proces prebudený.

Možnosť spustiť proces s právami administrátora výrazne ovplyvnil implementáciu mechanizmu spúšťania procesu. Operačný systém Windows neposkytuje rozhranie pre spustenie procesu s právami administrátora (vyvolanie dialógu UAC) súčasne v spiacom stave z procesu bez zvýšených práv. Pretože používateľské rozhranie beží bez administrátorských práv a služba nie je vhodná pre vytváranie procesov (vykonáva sa v systémovom procese), bol navrhnutý samostatný program. Spúšťanie procesov musí byť čo najrýchlejšie, a preto je spúšťač procesov implementovaný v jazyku C. Komunikácia medzi používateľským rozhraním a spúšťačom procesov sa uskutočňuje cez pomenované rúry bez WCF abstrakcie. GUI pomocou komunikácie so spúšťačom dokáže zobudiť spiaci proces, aplikovať obmedzenia zdrojov a v prípade neočakávanej chyby terminovať proces. Spúšťač procesov sa spúšťa pri každom štarte nového procesu, ktorý používateľ vytvára s právami administrátora a automaticky ukončí ak je proces prebudený resp. ak dostane signál s informáciami o prebudení procesu.

5.5.5 Analýza činnosti

Monitorovanie aktivity procesov je dobrým spôsobom pre analýzu chovania škodlivého softvéru. Jadro pieskoviska zaznamenáva súborovú a registrovú činnosť pre každé logické pieskovisko. V zostavenom ovládači sa logujú všetky súborové operácie okrem čítania a enumerácie. Čítanie súborov nie je kritická operácia pri vystavenom útoku a pri množstve čítaných súborov a knižníc potrebných už len pre štart aplikácie, by prehliadanie záznamov pôsobilo zmätene. Podobne pri registroch, ignoruje sa čítanie a enumerácia kľúčov a hodnôt. Používateľské rozhranie obsahuje nástroj pre prehliadanie záznamov. Jednotlivé záznamy sú rozdelené podľa procesov. V záujme prehľadnosti sa zobrazujú iba procesy spustené resp. ukončené v aktuálnom sedení. Používateľská aplikácia obsahuje databázu kritických ciest v rámci súborov a registrov. Databáza obsahuje miesta týkajúce sa spúšťania programov pri štarte, systémových knižníc atď. Ak proces z pieskoviska resp. argument súborovej alebo registrovej operácie participuje so záznamom z databázy, uvedený záznam bude obsahovať upozornenie vo forme ikony s výkričníkom.

6 Zhodnotenie riešenia

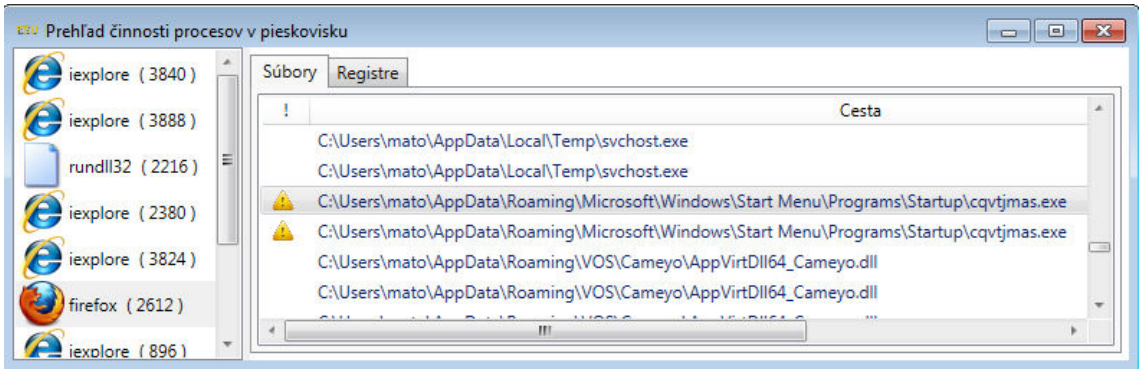
Kľúčovým cieľom riešenia je predísť infikovaniu a poškodeniu hostiteľského operačného systému. Izolovaný proces v žiadnom prípade nesmie modifikovať OS mimo vytvoreného pieskoviska. V tejto kapitole je testovaná bezpečnosť navrhovaného pieskoviska a vplyv kontrolovaného vykonávania na výkon resp. potrebné režijné náklady. Okrem testovania bezpečnosti a výkonu obsahuje posledná časť subjektívne hodnotenie riešenia nadobudnuté počas práce s aplikáciou.

6.1 Bezpečnosť riešenia

Overenie bezpečnosti riešenia pozostáva zo spustenia testovacích programov v prázdnom pieskovisku. Predtým sa vytvorí obraz celého súborového systému a databázy registrov. Po testovaní sa pieskovisko odstráni vrátane všetkých súborov a vytvorených kľúčov v registroch. Následne sa porovná uložený obraz súborového systému a databázy registrov s aktuálnym stavom. V ideálnom prípade nedôjde k žiadnej zmene. Pre snímkovanie systému sa použije program *System Explorer*. Pieskovisko bolo testované vo virtuálnom stroji s aktualizovaným operačným systémom Windows 7 v 32 bitovej verzii. V pieskovisku nebola aktivovaná kontrola zdrojov, obmedzenia systémových nastavení a sieťová izolácia.

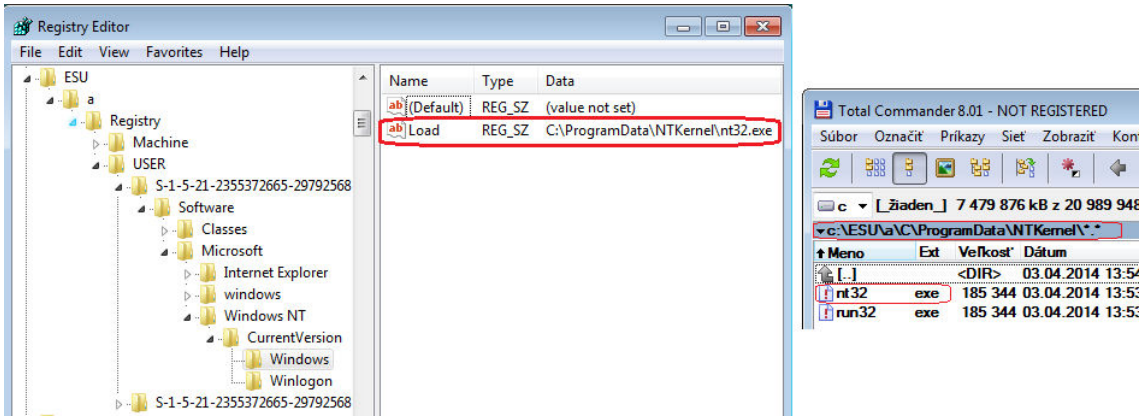
Test prebiehal v internetovom prehliadači Internet Explorer 8 (IE8). Päť rokov stará verzia bola vybratá z dôvodu väčšej zraniteľnosti. Počas testu bol vypnutý antivírusový program MS Security Essentials a nastavená minimálna ochrana v internetovom prehliadači. Pri testovaní sa prehliadali nebezpečné stránky zo zoznamu na adrese malwaredomainlist.com. Na stránkach sa zámerne klikalo na podozrivé odkazy. Počas tohto testovania boli spustené (úmyselne aj neúmyselne) škodlivé aplikácie, ktoré samovoľne spúšťali predvolený internetový prehliadač Firefox 15 a príkazový riadok. Testovalo sa 30 minút. Aj po ukončení IE8 v pieskovisku naďalej bežal trójskym koňom v pozadí spustený neaktuálny Firefox, ktoré jadro po ukončení práce s pieskoviskom a ukončení GUI terminovalo. Následne sa skontrolovala aktivita v pieskovisku cez grafické rozhranie pieskoviska. Na Obr. 18 je vidieť ako navrhnuté pieskovisko zabránilo vytvoreniu spúšťačov nebezpečných programov v umiestneniach pre automatický štart. Okrem toho, podľa množstva záznamov v prehliadači GUI s príznakom zápisu a odstránenia, škodlivý softvér hľadal a zároveň modifikoval všetky DLL a spustiteľné súbory v systéme. Jadro pieskoviska v snahe zabrániť modifikácii

originálnych súborov, kopírovalo a zapisovalo súbory do domény pieskoviska. V dôsledku toho, pieskovisko po teste obsahovalo až 900 MB dát.



Obr. 18 Aktivita v pieskovisku po bezpečnostnom teste

Ďalej sa škodlivý softvér snažil pridať hodnoty do troch rozličných kľúčov pre štart programov. Príkladom je pokus o načítanie programu pri prihlásení používateľa prostredníctvom hodnoty `Load` v kľúči `HKCU\Software\Microsoft\Windows NT\CurrentVersion\Windows`. Úspešné izolovanie hodnoty aj súborov je možné vidieť cez editor registrov a program Total Commander na Obr. 19.



Obr. 19 Izolovanie škodlivého softvéru v pieskovisku registrov a súborov

Po ukončení práce s pieskoviskom nasledovalo vytvorenie snímku všetkých súborov a registrov pre porovnanie s pôvodným stavom. V rámci registrov boli mimo pieskoviska zmenené kľúče a hodnoty týkajúce sa štartu aplikácií a interných technológií ako je Prefetch a SuperFetch. Išlo napr. o kľúč `HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\UserAssist`, ktorý uchováva informácie o často otváraných súboroch a odkazoch. S tým súvisela aj modifikácia súboru mimo pieskoviska. V ntuser.dat je uložený celý strom kľúča HKCU. Okrem tohto súboru, boli mimo pieskoviska zapísané štandardné záznamy vytvorené denníkom udalostí systému Windows a súbory monitora výkonu

nesúvisiace s testovanými aplikáciami. Nebol spozorovaný žiaden podozrivý súbor mimo pieskoviska.

Slabinou navrhnutého riešenia je neošetrená komunikácia medzi procesmi. Komunikáciu je možné buď úplne eliminovať voľbou v nastaveniach pieskoviska (môže ovplyvniť vykonávanie) alebo ponechať povolenú, čo prinesie možné bezpečnostné riziká. Windows neposkytuje rozhranie pre filtrovanie komunikácie. Jedným z riešení by bolo použiť techniku modifikácie SSDT tabuľky. Tým by došlo k porušeniu portability medzi 32 a 64 bitovými verziami a bolo by potrebné udržiavať v 64 bitovej verzii kód pre vypnutie KPP ochrany. Iným riešením by bolo použitie prerušení v režime používateľa. Tieto prerušenia však nie sú také bezpečné. Služba pieskoviska obsahuje rozhranie pripravené pre implementáciu používateľských prerušení na ošetrenie komunikácie medzi procesmi. Okrem komunikácie medzi procesmi by do budúcnosti bolo dobré pridať obmedzenie pre maximálnu veľkosť pieskoviska, aby nedochádzalo k podobným situáciám ako pri teste. Napriek spomenutým nedostatkom navrhnuté pieskovisko v súčasnom stave dokáže zabrániť mnohým útokom.

6.2 Výkonnostný deficit

Pieskovisko prerušuje systémové volania resp. filtruje operácie na súboroch a registroch. Hlavné režijné náklady sú spojené s presmerovaním požiadaviek do pieskoviska, čo zahrňuje identifikáciu procesu, alokovanie pamäti a kontextov, konštrukciu reťazcov, hľadanie záznamov v tabuľke odstránených položiek a najmä systémové volania súvisiace s metódou kopírovania pri zápise, kde patrí overenie existencie súborov a kľúčov, kopírovanie súborov, kľúčov, hodnôt atď. V tejto časti sú zmerané dodatočné náklady jednotlivých operácií a taktiež porovnanie času štartu procesov v pieskovisku a v natívnom, hostiteľskom prostredí. Testovanie prebehlo v 32 bitovom operačnom systéme Windows 7 nainštalovanom vo virtuálnom stroji softvéru VirtualBox 4.3. Virtuálny stroj mal pridelených 2 GB operačnej pamäte, zapnutú podporu hardvérovej virtualizácie a dostupné dve jadrá procesora Intel Core i5-4440 s maximálnou frekvenciou 3,10 GHz. Operačný systém bol spustený v testovacom režime. Pri všetkých testoch bol aktívny antivírus MS Security Essentials. Ovládač bol preložený bez optimalizácií a s ladiacimi symbolmi a výpismi.

Celkový počet strojových cyklov jednotlivých operácií v jadre sa meral od vstupu do spätného volania po jeho výstup pomocou funkcie *QueryPerformanceCounter*, ktorá s vysokou presnosťou počíta časové značky resp. meria časový interval. Postup pri

testovaní bol nasledovný. Bolo vytvorené nové, prázdne logické pieskovisko, v ktorom sa otvoril program „Poznámkový blok“. V ponuke Súbor/Otvoriť sa postupne v dialógu pre výber súboru preklikávalo z umiestnenia C:\ do umiestnenia C:\Users\mato\Desktop, kde sa vybral textový súbor o veľkosti 100 bajtov. Následne v poznámkovom bloku bol dopísaný jeden znak a súbor sa uložil. Opäť bol vyvolaný dialóg pre otvorenie súboru. Z umiestnenia C:\Users\mato\Desktop sa okamžite prešlo do zväzku C:\, kde sa premenoval priamo v dialógu textový súbor o veľkosti 12 bajtov. Premenovaný súbor bol načítaný do editora. Nakoniec bol poznámkový blok ukončený. Počet strojových cyklov jednotlivých operácií je v Tab. 3. Z tabuľky možno napríklad vyčítať, že operácia otvorenia/vytvorenia súboru je najčastejšia, ale jednoznačne najdlhšia je operácia enumerácie adresárov. Naproti tomu enumerácia kľúčov a hodnôt nie je v tomto príklade časovo náročná, pretože ako bolo spomenuté v časti 4.3.5, volanie vráti vždy iba jeden záznam. Zaujímavosťou je vysoký počet operácií u dopytovaní hodnoty.

Tab. 3 Čas a počet strojových cyklov pri vykonaní operácií v jadre pieskoviska

Operácia	Počet operácií	Strojových cyklov	Priemerný počet strojových cyklov	Celkový čas [ms]
Pre-vytvoriť	2 546	2 999 709	1 178	1 094
Post-vytvoriť	547	42 652	78	15
Nastaviť informácie	15	6 898	456	3
Enumerácia	172	5 654 296	32 874	2 065
Otvoriť kľúč	9 080	1 153 549	127	416
Vytvoriť kľúč	245	114 778	468	41
Nastaviť hodnotu	178	273 426	1 536	99
Dopytovať kľúč	3 753	423 862	113	153
Dopytovať hodnotu	11 488	271 781	24	94
Enumerácia kľúča	657	14 004	21	5
Enumerácia hodnoty	282	6 583	23	2
Odstránenie kľúča	0	0	0	0
Odstránenie hodnoty	12	686	57	0, 246

Tabuľka neobsahuje percentuálne porovnanie s natívnym vykonávaním, pretože bez prerušenia volania metódou modifikácie SSDT tabuľky, nie je možné v jadre priamo odmerať čas natívneho volania. Presnejší obraz pre porovnanie rýchlosti vykonávania v pieskovisku a u hostiteľa možno získať testovaním štartu programov a ich doby vykonávania. Pre tieto účely bol vytvorený pomocný program štartujúci procesy. Program meria čas od štartu procesu po jeho inicializáciu v prípade testovania rýchlosti štartu programu. Pre meranie celkového času vykonávania pomocný program meria čas dovtedy, pokiaľ proces neukončí svoju činnosť. Prvé spustenie programov je dlhšie a označuje sa ako studený štart. Opakované štartovanie vďaka načítaným knižniciam prebieha rýchlejšie. Meraný bol studený aj opakovaný štart programov. Testovaný bol štart internetového prehliadača Firefox 28, komprimačného programu WinRAR 5, prehliadača PDF súborov Foxit Reader 6 a klienta pre prenos súborov Emule 0.5. Pri meraní času studeného štartu bol OS po každom pokuse reštartovaný. Výsledky meraní sú v Tab. 4. Každý pokus bol zopakovaný päť krát. Priemerné náklady pri prvom štarte predstavujú 8%, pri opakovanom štarte 17%. Výkonnostný deficit závisí od toho aké a koľko knižníc programy pri štarte používajú. Napr. v prípade programu WinRAR, väčšinu nákladov tvorila enumerácia súborov vykonávaná vždy pri štarte. Menší percentuálny podiel pri studenej inicializácii súvisí s nárastom času pri čítaní knižníc z pevného disku.

Tab. 4 Studený a opakovaný štart procesov v pieskovisku a u hostiteľa

	Prvý štart u hostiteľa [ms]	Prvý štart v piesk. [ms]	Op. štart u hostiteľa [ms]	Op. štart v pieskovisku [ms]
Firefox	3060	3423 (11%)	106	130 (18%)
Emule	2134	2219 (4%)	476	545 (13%)
Winrar	959	1124 (15%)	148	219 (32%)
FR 6	3937	4045 (3%)	30	31 (5%)

Celkový čas vykonávania bol meraný v dvoch programoch. Prehliadač obrázkov IrfanView hromadne konvertoval 280 obrázkov rôznej veľkosti (dokopy 116 MB) typu PNG na formát JPG. Zdrojové a cieľové umiestnenie obrázkov bolo počas testov procesu vykonávaného v pieskovisku menené medzi hostiteľom, pieskoviskom a zväzkami. Zmena nemala podstatný vplyv na čas vykonávania. Druhý test pozostával z práce v komprimačnom programe WinRAR. Po štarte programu test simuloval jeho používanie. Uskutočnil sa postupný prechod z adresára C:\Users\mato\Desktop do

koreňa C:\ a následne opätovný postupný prechod do štartovacieho adresára. Vybral sa adresár veľkosti 90MB obsahujúci stiahnutú internetovú stránku s 3250 súbormi a spustila sa komprimácia. Prechod medzi jednotlivými úrovňami súborového systému bol automatizovaný a z dôvodu čo najreálnejšej simulácie prerušovaný sekundovými intervalmi. Pred spustením komprimácie sa zaznamenal medzičas. Celkový výsledný čas je tvorený súčtom medzičasu a času komprimácie. Medzičas charakterizuje používanie aplikácie z hľadiska jeho odozvy a tiež používanie registrov. Bolo zaznamenaných 12 600 filtrovaných registrových operácií. Druhá časť skôr predstavuje syntetický test súborovej virtualizácie. Bolo vykonaných desať meraní, z ktorých sa vypočítal aritmetický priemer. Výsledné časy namerané počas natívneho a kontrolovaného vykonávania sú v Tab. 5. Medzičasy pri druhom teste sú uvedené v zátvorkách.

Tab. 5 Celkový čas vykonávania programu v pieskovisku a u hostiteľa

	Hostiteľ [s]	Pieskovisko [s]	Réžia
IrfanView	5,363	6,086	12%
WinRAR	17,824 (8,886)	18,847 (9,002)	5% (1%)

Obsadenie pamäte pri načítanom jadre predstavuje približne 800 KB pamäte. S vytvorenými šiestimi prázdnymi logickými pieskoviskami obsadenie stúpne o 3 KB. Najvýznamnejší vplyv na ďalší nárast ma počet záznamov v tabuľke odstránených položiek. Napr. ak by tabuľka obsahovala 50 000 vymazaných záznamov s maximálnou dĺžkou mena súboru, využitá pamäť by stúpla o 25 MB. Takto veľká tabuľka nie je pri použití pieskoviska očakávaná, avšak pri súčasných veľkostiach operačnej pamäte to nie je extrémne číslo.

Štandardné používanie programov v pieskovisku minimálne degraduje čas ich vykonávania. Vyššie režijné náklady sú pri programoch s intenzívnymi V/V operáciami. Všetky ostatné systémové volania sa vykonávajú natívne, čo je výhoda v porovnaní s virtuálnymi strojmi na úrovni hardvéru. Celkový výkonnostný pokles záleží od druhu aplikácie a môže sa pohybovať od 5% po 20%. S nástupom rýchlych diskov bez pohyblivých častí (SSD), čas potrebný najmä pri overovaní existencie a kopírovaní súborov rapídne klesá. Napriek tomu, pre zlepšenie celkového výkonu možno do jadra pieskoviska napr. pridať pomocnú vyrovnávajúcu pamäť pre udržiavanie mien súborov z pieskoviska, zlepšiť implementáciu enumerácie súborov alebo optimalizovať prístup

k pomocným tabuľkám. S ohľadom na uvedené testy však možno konštatovať, že navrhnuté pieskovisko je pre štandardné použitie dostatočne rýchle.

6.3 Práca s aplikáciou

Vytvorený softvér testovalo niekoľko používateľov. Zväčša pozitívne pôsobil celkový vzhľad a spôsob práce s aplikáciou. Vytváranie, odstraňovanie pieskovísk resp. spúšťanie a ukončovanie procesov je animované a sprevádzané jednoduchými vizuálnymi efektmi. Používatelia neznalí aplikácie aj bez používateľskej príručky po chvíli program vedeli v predvolenom stave používať. Vďaka spätnej väzbe boli opravené viaceré chyby týkajúce sa prevažne stability a vstupov používateľskej aplikácie. Napriek tomu, negatívom v niektorých prípadoch bola nekonzistencia procesov v pieskovisku. Občas sa stávalo, že aplikácia vypísala dialóg o chybe alebo sa nesprávala ideálne. Zabezpečenie čo najväčšej konzistencie procesov a ich stabilné vykonávanie je dôležitou výzvou v ďalšom vylepšení riešenia.

Rozsah používateľov pre testovanie softvéru bol značne obmedzený. Pohodlne testovať mohli iba osoby s operačným systémom v 32 bitovej verzii. Aplikácie vykonávané v režime používateľa t.j. služba a grafické rozhranie so všetkými knižnicami po preložení do 64 bitovej architektúry možno použiť bez ťažkostí. Problémom je jadro pieskoviska. Napriek tomu, že ovládač je pre 64 bitovú verziu pripravený a bol testovaný vo Windows Vista x64, nie je možné ho jednoducho zaviesť do systému. Všetky 64 bitové ovládače musia byť podpísané u certifikačnej autority. Certifikácia ovládačov je časovo a finančne náročný proces. V súčasnom stave, kedy ovládač neobsahuje filtrovanie komunikácie medzi procesmi, sieťovú virtualizáciu a nie sú ošetrené a intenzívne otestované všetky verzie OS, certifikácia neprichádza do úvahy. Spôsob ako použiť riešenie v 64 bitových OS nie je triviálny. Základný postup je spomenutý v používateľskej príručke.

7 Záver

Zámerom tejto práce bolo navrhnúť systémové pieskovisko v operačnom systéme Microsoft Windows. Vykonávanie aplikácií v pieskovisku by malo byť čo najbezpečnejšie, dostatočne rýchle a konzistentné. Preskúmaním vzťahu medzi virtualizáciou a konceptom systémových pieskovísk došlo k výberu vhodného spôsobu pre umiestnenie virtualizačnej vrstvy v systéme. Navrhnuté pieskovisko je z architektonického hľadiska virtuálny stroj na úrovni operačného systému. Analýza existujúcich riešení poukázala na základné princípy systémových pieskovísk a ich hlavné problémy ako sú podpora najnovších OS, príliš reštrikčné a netransparentné správanie a v niektorých prípadoch náročná konfigurácia a ovládanie.

Vytvorená práca ukázala ako je možné implementovať systémové pieskovisko pomocou moderných technológií. Systémové volania sú filtrované v privilegovanom režime jadra. Súborové operácie sú kontrolované vďaka vytvorenému filtru súborového systému označovaného ako minifilter. Prístup do registrov je kompletne ošetrený vrstvou filtra registrových volaní. Algoritmus presmerovania súborových operácií je založený na metóde kopírovania pri súboroch otvorených na zápis, ktorá výhodne umožňuje čítať súbory hostiteľského systému a zakazuje jeho modifikáciu mimo povolených častí. Naopak, nevýhodou metódy je možné plytvanie diskovým priestorom. Algoritmus kontroly registrových systémových volaní pozostáva z metódy vytvárania kľúčov a hodnôt v pieskovisku až pri zápise hodnôt resp. pri vytváraní kľúčov. Uvedený princíp je implementovaný efektívnym spôsobom bez zbytočných nákladov. Celé riešenie bolo navrhnuté s možnosťou vytvárania viacerých nezávislých pieskovísk a samostatnej konfigurácie. Okrem základných častí systémového pieskoviska práca obsahuje metódy, ako obmedziť systémové zdroje operačného systému, jeho nastavenia a prístup na sieť prostredníctvom implementácie modulu vo forme služby OS Windows. Nakoniec bola navrhnutá grafická aplikácia pre jednoduché a intuitívne ovládanie pieskovísk. Využili sa dostupné moderné technológie vďaka čomu môže byť vizuálne atraktívna. V nadväznosti na tieto tri produkty bolo poukázané ako môžu spolu komunikovať, a ako celok vytvoriť bezpečné prostredie pre vykonávanie programov.

Do výsledného riešenia sa nepodarilo implementovať ošetrenie komunikácie medzi procesmi, čo je hlavný nedostatok práce. Časť komunikácie je možné buď

zakázať alebo povoliť. Ďalej by bolo potrebné vylepšiť sieťovú izolácie hlbšou integráciou do pieskoviska a sofistikovanejším riešením. Priestor pre zlepšenie sa dá nájsť aj pri implementácii časovo náročných operácií napr. u enumerácie adresárov. Tvorba ovládača, filtra súborového systému je neľahká úloha. Ako ukázali testy, potrebné je popracovať na stabilite riešenia a ošetrení všetkých aspektov, ktoré môžu nastať pri používaní ľubovoľnej aplikácie v pieskovisku, čo vyžaduje dlhodobé testovanie a vývoj.

Systémové pieskovisko nemá byť ambíciou a ani nie je náhradou antivírusových programov. Aplikácia sa osvedčí predovšetkým ako ich doplnok. Napriek spomenutým nedostatkom je výsledný produkt vhodný pre nenáročné používanie a diplomová práca dobrým začiatkom pre vstup do problematiky systémových pieskovísk v operačných systémoch Windows.

Zoznam použitej literatúry

[1] HOOPES, John: Virtualization for security including sandboxing, disaster recovery, high availability, forensic analysis, and honeypotting. Burlington, MA: Syngress Pub., 2009. 357 s. ISBN 978-1-59749-305-5

[2] WAHBE, Robert. et al: Efficient software-based fault isolation. In: SIGOPS Oper. Syst. Rev. 1993, roč. 27, č. 5, s. 203–216.

[3] SMITH, James Edward - NAIR, Ravi: Virtual Machines: Versatile Platforms for Systems and Processes. San Francisco: Elsevier, 2005. 662 s. ISBN 1-55860-910-5.

[4] BARRETT, Diane - KIPPER, Gregory: Virtualization and Forensics: A Digital Forensic Investigator's Guide to Virtual Environments. Burlington: Syngress, 2010. 274 s. ISBN 978-1-59749-557-8.

[5] MOYA DEL BARRIO, Victor: Study of the techniques for emulation programming [online]. Barcelona: Barcelona School of Informatics, FIB, 2001. [cit. 2013-04-12]. Dostupné na internete:<http://personals.ac.upc.edu/vmoya/docs/emuprog.pdf>

[6] CALZOLARI, Federico: High availability using virtualization: Dizertačná práca. Pisa: University of Pisa, 2009. 83 s.

[7] NAKAJIMA, Jun. - MALLICK, Asit: Hybrid-Virtualization - Enhanced Virtualization for Linux. In: Linux Symposium, 2007, č. 2, s. 87-96.

[8] CHAUDHARY, Vipin et al: A Comparison of Virtualization Technologies for HPC. In: 22nd International Conference on Advanced Information Networking and Applications, Buffalo: IEEE computer society, 2008. s. 861–868. ISBN 978-0-7695-3095-6.

[9] YU, Yang: Os-level virtualization and its applications: Dizertačná práca. Stony Brook, NY, USA: State University of New York at Stony Brook, 2007. 120 s.

[10] SHAN, Zhiyong et al: Facilitating inter-application interactions for OS-level virtualization. In: SIGPLAN Not. roč. 47, 2012, č. 7, s. 75–86.

[11] LAADAN, Oren - NIEH, Jason: Operating system virtualization: practice and experience. In: Proceedings of the 3rd Annual Haifa Experimental Systems Conference, New York, NY, USA: ACM, 2010. s. 1–12. ISBN: 978-1-60558-908-4

[12] WHITE, Joshua - PILBEAM, Adam: A Survey of Virtualization Technologies With Performance Testing [online]. 2010. [cit. 2013-04-14]. Dostupné na internete: < http://arxiv.org/ftp/arxiv/papers/1010/1010.3233.pdf >.

[13] KAMPERT, Paulus: Taxonomy of virtualization Technologies: Diplomová práca. Delft: Delft University of Technology Faculty of Systems Engineering Policy Analysis & Management, 2010. 93 s.

[14] KUSNETZKY, Dan: Virtualization a manager's guide. Sebastopol, CA: O'Reilly Media, Inc., 2011. 58 s. ISBN 978-1-449-30645-8.

[15] KAMP, Poul-Henning - WATSON, Robert N.M: Jails: Confining the omnipotent root. In: Proceedings of the 2nd International SANE Conference, Maastricht. 2000. s. 116-127.

[16] SANDBOXIE HOLDINGS LLC: Sandboxie - Frequently Asked Questions [online]. [cit. 2013-03-03]. Dostupné na internete: <http://www.sandboxie.com/index.php?FrequentlyAskedQuestions>.

[17] SINGH, Jasmet. et al: An Application Sandbox Model based on Partial Virtualization of Hard-Disk and a Possible Windows Implementation. In: International Journal of Computer Applications. roč. 7, 2012. č. 57. s. 16-21.

[18] KASAMPALIS, Sakis: Copy On Write Based File Systems Performance Analysis And Implementation: Diplomová práca. Kongens Lyngby: Technical University of Denmark, Department Of Informatics, 2010. 83 s.

[19] APAP, Frank. et al: Detecting Malicious Software by Monitoring Anomalous Windows Registry Accesses. In: Recent Advances in Intrusion Detection: Lecture Notes in Computer Science, Berlin: Springer Berlin Heidelberg, 2002. s. 36–53. ISBN 978-3-540-00020-4.

[20] HONEYCUTT, Jerry: Microsoft Windows Registry Guide. Redmond: Microsoft Press, 2005. 1327 s. ISBN 9780735637351.

[21] MICROSOFT: Synchronization Objects [online]. 2006. [2013-11-16] [cit. 2013-12-05]. Dostupné na internete: <http://msdn.microsoft.com/en-us/library/windows/desktop/ms686364%28v=vs.85%29.aspx>.

[22] MICROSOFT: Interprocess Communicatios [online]. 2006. [2013-11-16] [cit. 2013-12-05]. Dostupné na internete: <http://msdn.microsoft.com/en-us/library/windows/desktop/aa365574%28v=vs.85%29.aspx>.

[23] CHANG, Fangze. et al: Secure, User-level Resource-Constrained Sandboxing [online].1999. New York: New York University Department of Computer Science, Courant Institute of Mathematical Science. Dostupné na internete: < http://oai.dtic.mil/oai/oai?verb=getRecord&metadataPrefix=html&identifier=ADA439735>.

[24] MICROSOFT: Microsoft Portable Executable and Common Object File Format Specification [online]. 1999. [2013-02-06] [cit. 2013-12-05]. Dostupné na internete: <http://msdn.microsoft.com/en-us/library/windows/hardware/gg463119.aspx>.

[25] HONIG, Andrew: Practical Malware Analysis. San Francisco: No Starch Press, 2012. 802 s. ISBN 9781593274306.

[26] STEVENSON, Larry - ALTHOLZ, Nancy: Rootkits For Dummies. Indianapolis: Wiley Publishing, Inc., 2006. 380 s. ISBN 978-0-470-10183-4.

[27] IVANOV, Ivo: API hooking revealed - CodeProject. [online]. 2002. [cit. 2013-12-05]. Dostupné na internete: <http://www.codeproject.com/Articles/2082/API-hooking-revealed>.

[28] BERDAJS, Jan - BOSNIĆ, Zoran: Extending applications using an advanced approach to DLL injection and API hooking. In: Software: Practice and Experience. roč. 7, 2010, č. 40, s. 567–584.

[29] HUNT, Galen - BRUBACHER, Doug: Detours: Binary Interception of Win32 Functions. In: Proceedings of the 3rd Conference on USENIX Windows NT Symposium - Volume 3, Berkeley, CA, USA: USENIX Association, 1999. s. 14–23. Dostupné na internete: <http://research.microsoft.com/apps/pubs/default.aspx?id=68568>.

[30] HUSSE, Christoph: EasyHook - The reinvention of Windows API Hooking. In [online]. 2006. [2013-01-24] [cit. 2013-12-05]. Dostupné na internete: <http://easyhook.codeplex.com/>.

[31] HOGLUND, Greg - BUTLER, James: Rootkits: Subverting the Windows Kernel. Upper Saddle River, NJ: Addison-Wesley Professional, 2006. 354 s. ISBN 0-321-29431-9.

[32] MICROSOFT: I/O request packets (Windows Drivers) [online]. 2006. [2013-11-16] [cit. 2013-12-05]. Dostupné na internete: <http://msdn.microsoft.com/en-us/library/windows/hardware/hh439638%28v=vs.85%29.aspx>.

[33] MICROSOFT: Choosing a driver model [online]. 2006. [2013-12-16] [cit. 2014-02-01]. Dostupné na internete: <http://msdn.microsoft.com/en-us/library/windows/hardware/ff554652%28v=vs.85%29.aspx>

[34] MICROSOFT: Writing Preoperation and Postoperation Callback Routines [online]. 2006. [2014-01-29]. Dostupné na internete: < http://msdn.microsoft.com/en-us/library/windows/hardware/ff557334%28v=vs.85%29.aspx >

[35] MICROSOFT: Job Objects [online]. 2006. [2014-02-16] Dostupné na internete: <http://msdn.microsoft.com/en-us/library/windows/desktop/ms684161%28v=vs.85%29.aspx>

[36] ANDERSON, Chris: Essential Windows Presentation Foundation (WPF). Boston: Addison-Wesley Professional, 2007. 772 s. ISBN 9780132701617.

Prílohy

Príloha A: DVD médium – diplomová práca v elektronickej podobe, prílohy v elektronickej podobe, zdrojové kódy aplikácie, preložené kódy aplikácie a dokumentácia zdrojových kódov.

Príloha B: Používateľská príručka

Príloha C: Systémová príručka

Príloha D: Spracovaný článok