Univerzita Komenského v Bratislave

Fakulta Matematiky, Fyziky a Informatiky

Trusted Path Execution v prostredí MS Windows

2010

Juraj Danko

Univerzita Komenského v Bratislave

Fakulta Matematiky, Fyziky a Informatiky

Trusted Path Execution v prostredí MS Windows

Bakalárska práca

Študijný program: Informatika
Študijný odbor: 9.2.1 Informatika
Školiace pracovisko: Katedra Informatiky
Školiteľ: Mgr. Peter Košinár

Bratislava, 2010

Juraj Danko

Čestne prehlasujem, že som túto bakalársku prácu vypracoval samostatne s použitím citovaných zdrojov.

.....

Ďakujem vedúcemu svojej bakalárskej práce, Mgr. Petrovi Košinárovi, za zaujímavú tému a efektívnu snahu pomôcť pri jej realizácii. Tiež by som chcel poďakovať RNDr. Michalovi Foriškovi PhD. za vytvorenie šablóny a ďalších materiálov, ktoré mi uľahčili výrobu tejto práce.

Abstrakt

Autor:	Juraj Danko
Názov bakalárskej práce:	Trusted Path Execution v prostredí MS Windows
Škola:	Univerzita Komenského v Bratislave
Fakulta:	Fakulta Matematiky, Fyziky a Informatiky
Katedra:	Katedra Informatiky
Vedúci bakalárskej práce:	Mgr. Peter Košinár
Rozsah práce:	51 strán
Bratislava, jún 2009	

Z dôvodu absencie riešenia umožňujúceho efektívne riešenie Trusted Path Execution, v konkrétnej interpretácií kontroly spúšťania aplikácií, v prostredí stredných a väčších firemných sietí, sa vynára zaujímavá možnosť prispieť vypracovaním konceptu takéhoto riešenia. Pokúsili sme sa preto analyzovať potreby a špecifiká využitia v takejto oblasti a navrhnúť štruktúru systému, ktorá by bola veľmi pružná a modulárna. Práve modulárnosť je totiž nevyhnutná z dôvodu špecifických požiadaviek každého z väčších subjektov; takto dosiahnutá dynamika riešenia umožňuje ľahké, rýchle a relatívne lacné úpravy podľa vyšpecifikovaných požiadaviek. Ďalšími nepochybnými prioritami návrhu a vývoja sú i výkon aplikácie či jej stabilita. Pri analýze spúšťania programov je otázka rýchlosti totiž esenciálnou; jej neefektívnosť by mohla viesť k rapídnym problémom s výkonom počítača ako takého. Systematický postup práce a dôkladná analýza problému viedli k súčasnému stavu projektu. Verím preto, že bude použiteľným konceptom pre každého, kto sa v budúcnosti pokúsi podobný systém navrhnúť či vyvinúť.

Kľúčové slová: Trusted Path Execution, Whitelist, Windows API.

Abstract

Author:	Juraj Danko
Title:	Trusted Path Execution in MS Windows
University:	Comenius University in Bratislava
Faculty:	Faculty of Mathematics, Physics and Informatics
Department:	Department of Computer Science
Advisor:	Mgr. Peter Košinár
Page count:	51 pages

Bratislava, 2009

Because of absence of a solution of the Trusted Path Execution - application launching control - problem in the segment of middle and bigger companies, it was interesting to try to develop an approaching application. We've tried to analyze needs and specific features of the mentioned segment and to intend a quite modular and dynamic structure of such a system. Especially the modularity of the system should be a must-have as each of affected subjects could have its own client-specific needs; such a dynamics we gain allows facile, fast and hopefully cheap modifications due to the demands specified. Further important priorities of the intended system should be the performance and stability of the final application product. When it comes to application launching, performance of such a system would be essential as its ineffectiveness could lead to serious computer performance problems. Therefore, the developed solution is reacting to a deep analysis of a problem. Hopefully it would be a useful concept for everyone trying to propose or develop such a system.

Keywords: Trusted Path Execution, Whitelist, Windows API.

Predhovor

Problematika bezpečnosti počítačov a počítačových sietí sa v poslednom desaťročí stáva jedným z najdôležitejších elementov informatiky ako takej. Vzhľadom na nevyhnutnosť neustáleho vývoja v oblasti bezpečnosti je dôležité uplatňovať nové prístupy a postupy pri jej uplatňovaní. Verím, že táto práca bude užitočnou pomôckou pri ďalšom vývoji zabezpečovacích systémov pre osobné počítače, a to nie len ako motivácia, ale aj ako zbierka nápadov a riešení, ktoré by mohli pomôcť riešiť praktické problémy pri ich prípadnej implementácii.

Obsah

Zoznam obrázkov

Úvod

Táto práca vznikla na základe potreby praxe, kde sa dlhodobo ukazuje absencia komplexného riešenia problému spúšťania neznámych či nedôveryhodných aplikácií. Riešenie je potrebné ako pre koncové stanice priamo v spoločnosti, tak i pre používanie staníc mimo nej, a jeho potreba je podčiarknutá častým používaním administrátorských účtov ako užívateľských na mnohých počítačoch používajúcich operačné systémy Windows. Predovšetkým väčšie spoločnosti a korporácie potrebujú zabezpečiť svoje počítače pred inštaláciou nelegálnych, nevhodných alebo kolidujúcich aplikácií. Doterajšie využitie manuálneho kontrolovania obsahu počítača zamestnancami IT oddelenia (alebo rodičmi v prípade domácnosti) zjavne nevyhovuje moderným trendom automatizácie v sfére osobných počítačov.
V časti 1 sa pokúsime objasniť výber z širokého spektra možností riešenia problémov, ktoré nás pri návrhu riešenia postretnú. Je totiž potrebné zvoliť vhodnú pozíciu v štruktúre operačného systému, v tomto prípade sa obmedzíme na konkrétny systém Microsoft Windows XP. Neskôr porovnáme výhody a nevýhody user-mode a kernel-mode prístupu. Tiež je dôležité zvoliť správnu formu užívateľského a administračného rozhrania. Zaujímavou výzvou je i štruktúra databázy, ktorá musí byť prispôsobená momentálnym funkčným obmedzeniam dostupných databázových systémov; treba však spomenúť, že spomenuté parametre možno vďaka vysokej modularite systému neskôr ľahko zmeniť či doplniť. Zaujímavým problémom sa ukázala byť aj voľba komunikačných rozhraní medzi jednotlivými komponentmi systému. Do budúcnosti dôležitou je i myšlienka rozšíriteľnosti navrhovaného systému na ďalšie platformy; rozšírenie v rámci 32-bitovej platformy MS Windows by malo byť jednoduchšie, prekážky sa ukazujú pri prechode na 64-bitovú platformu. Vďaka vysokej modularite systému je reálne i rozšírenie na ďalšie platformy, a to vrátane operačných systémov Linux, Solaris či Mac OS; touto možnosťou sa však nebudeme podrobnejšie zaoberať.

Pri voľbe mnohých uvedených i ďalších parametrov a riešení som čerpal okrem odborných publikácií i z existujúcich riešení, ktoré sa pokúšajú riešiť niektoré čiastkové podproblémy problému, pre ktorý v tejto práci navrhneme riešenie. Je však potrebné povedať, že

i pri dlhom a namáhavom hľadaní sa mi nepodarilo objaviť produkt s obdobnou funkčnosťou a cieľovou skupinou vhodne nasaditeľný do podmienok praxe; verím preto, že sa túto medzeru v ponúkaných službách podarí vyplniť aj pomocou tejto práce.

Kapitola 1

Analýza dotknutých súčastí systému Windows XP a komunikácia

V tejto časti práce sa sústredíme na prostredie operačného systému a prvky v jeho štruktúre, ktoré viac či menej ovplyvňujú a obmedzujú vývoj aplikácie.

1.1 Windows API

Predmetom tejto sekcie bude uviesť zjednodušený pohľad na pre nás dôležité časti Windows API[1] systému Microsoft Windows XP a vysvetliť dôvody voľby driveru[2] ako prostriedku na detekciu a riadenie vytvárania procesov. Podľa [10], štruktúru API možno zjednodušene znázorniť diagramom 1.2, kde sú uvedené aj informácie o volaní a úlohách jednotlivých komponentov.

Procesory série x86 v súčasných verziách podporujú multitasking a rôzne mechanizmy bezpečnosti. Kód aplikácií môže byť vykonaný v jednej zo 4 existujúcich úrovní, Ring 0 až Ring 3. Systémy Windows vo verzii XP, Vista, 7, ako aj mnohé iné dnes používané operačné systémy, využívajú z tohto spektra iba 2 úrovne, a to Ring 0 a Ring 3. Ring 3 sa nazýva aj používateľská úroveň, sú v nej spúšťané aplikácie používateľa, zatiaľ čo na úrovni Ring 0, teda úrovni jadra, sú spúšťané procesy operačného systému a driverov. Z behu aplikácie v režime jadra plynú výhody i nevýhody; nepochybnou výhodou je možnosť robiť "všetko" - aplikácia môže priamo pristupovať ku ktorejkoľvek časti pamäte. Táto možnosť je však zároveň aj nevýhodou; takýto prístup a prípadné chyby v aplikácii totiž

[1] Prostredie programovania aplikácií, preklad z Application Programming Interface (angličtina)

[2] Driver možno preložiť ako ovládač (angličtina); z dôvodu zaužívania slova driver i na Slovensku budeme v ďalšom texte preferovať použitie slova driver, drivery (mn.č.)

môžu spôsobiť závažné chyby bežiacich aplikácií, ale aj operačného systému ako takého. Nebezpečenstvom je aj fakt, že na tejto úrovni sú výnimky ošetrené takzvaným "Blue Screen", modrou obrazovkou, ktorá znamená núdzové ukončenie operačného systému. Vyplývajúcou nevýhodou je aj nemožnosť použiť množstvo bežných knižníc, ktoré výnimky používajú; aplikácia z režimu jadra nemôže volať funkcie z používateľského režimu, a aj ak by ich volala, bolo by to z vyššie uvedených dôvodov veľmi nezodpovedné.

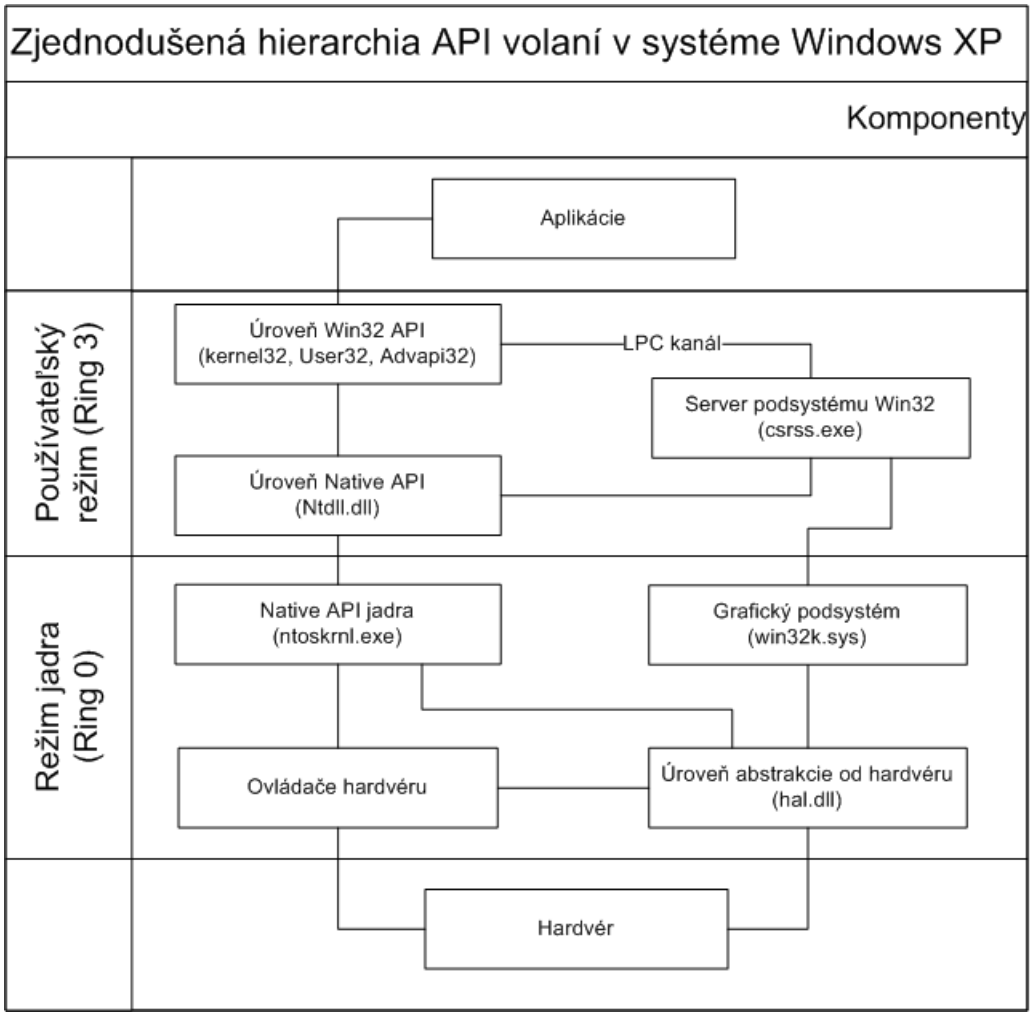
Na druhej strane, aplikácie bežiace v užívateľskom režime nemôžu priamo modifikovať stránkovanie pamäte, a teda nemajú možnosť prístupu do pamäte iných aplikácií inak ako cez API volania, čo by pre navrhovaný systém bol ťažko riešiteľný problém; takáto funkčnosť by navyše mohla byť užitočná aj pre ďalšie prípadné rozšírenia navrhovaného systému. Citujme presné vyjadrenie z literatúry [6] (str. 8): „Kernel objects must be manipulated by Windows APIs. There are no "back doors." This arrangement is consistent with the data abstraction principles of object-oriented programming, although Windows is not object oriented." Aplikácie spúšťané v používateľskom režime tiež nedokážu ovplyvniť prerušenia či Context Switching - zmenu bežiacej aplikácie. Keďže naším cieľom je vytvoriť systém, ktorý bude detegovať všetky pokusy o spustenie aplikácie a nechceme, aby bolo možné mu jednoducho podvrhnúť nepravdivé dáta, o voľbe režimu jadra pre našu aplikáciu rozhodol predovšetkým fakt, že aplikácia v používateľskom režime nedokáže priamo pristupovať k štruktúram aplikácií v režime jadra.

1.2 Procesy

Podľa [3] v systémoch Microsoft Windows XP a vyšších každý proces disponuje vlastným:

Virtuálnym adresným priestorom
Fyzickou pamäťou (Working set)
Access tokenom
Handle table pre objetky jadra Win32
Stack (zásobníkom)
Inštanciou top-level funkcie
Stavom časovača (Wait, Ready, Running, ...)
Prioritou
Prístupovým módom (User Mode alebo Kernel Mode)
Uloženým stavom CPU (ak práve nebeží)
Access tokenom (nepovinné)

Obr. 1.1: Štruktúra Windows API



Predmetom záujmu navrhovaného systému bude detegcia procesov vytvorených v používateľskom móde (User-Mode). Takéto zjednodušenie je možné z dôvodu, že ak by sa používateľ pokúsil spustiť program v režime jadra (kernel-mode), bolo by potrebné najprv nahrať driver, čo si vyžaduje spustenie aplikácie v používateľskom režime. Preto filtrovanie aplikácií spúšťaných v používateľskom režime pokrýva aplikácie spúšťané v systéme známym spôsobom.[3] Podľa [2] a [5], pre spustenie aplikácie je potrebné najprv aplikáciu otvoriť s príslušnými právami, následne nahrať jej obsah do pamäte, zvyčajne pamäte RAM. Tiež je potrebné nastaviť **Process Executive Object** a alokovať adresný priestor. Po namapovaní do adresného priestoru systém analyzuje zoznam externých funkcií z dynamických knižníc (DLL) a pre každú (aj vnorenú) knižnicu volá funkciu **LoadLibrary**. Tiež je potrebné nastaviť **Thread Executive Object** a alokovať zásobník pre primárne vlákno aplikácie. Následne je potrebné informovať **subsystém Win32** o novom procese spolu s parametrami jeho spustenia. Pre úspešné spustenie je potrebné, aby všetky uvedené rutiny boli vykonané správne. Podľa [3], podstatné časti spustenia aplikácie v používateľskom režime možno zhrnúť diagramom 1.2.

Z povahy a praktického využitia funkcie NtCreateProcess() vyplýva, že hookovať[4] túto funkciu by nebolo úplným riešením. Existujú totiž bežne využívané postupy vytvorenia procesu, ktoré túto funkciu obchádzajú; príkladom by mohla byť funkcia CreateProcess(). Preto podľa [2] sú vhodnými funkciami na hookovanie buď NtCreateFile() spolu s NtOpenFile() alebo NtCreateSection(), pretože ich pri vytváraní procesu nie je možné obísť. Hookovanie NtOpenFile() sa však ukazuje problematickým z dôvodu praktického; táto funkcia je totiž využívaná aj pri bežných vstupno-výstupných operáciách.

Funkcia NtCreateSection() je, ako sme už povedali, nevyhnutnou[5] pre vytvorenie procesu. Ak teda rozhodneme o povolení vytvorenia procesu, stačí ju nechať korektne vykonať svoju činnosť; v opačnom prípade aplikácia vráti hodnotu STATUS_ACCESS_DENIED a proces nebude vytvorený.

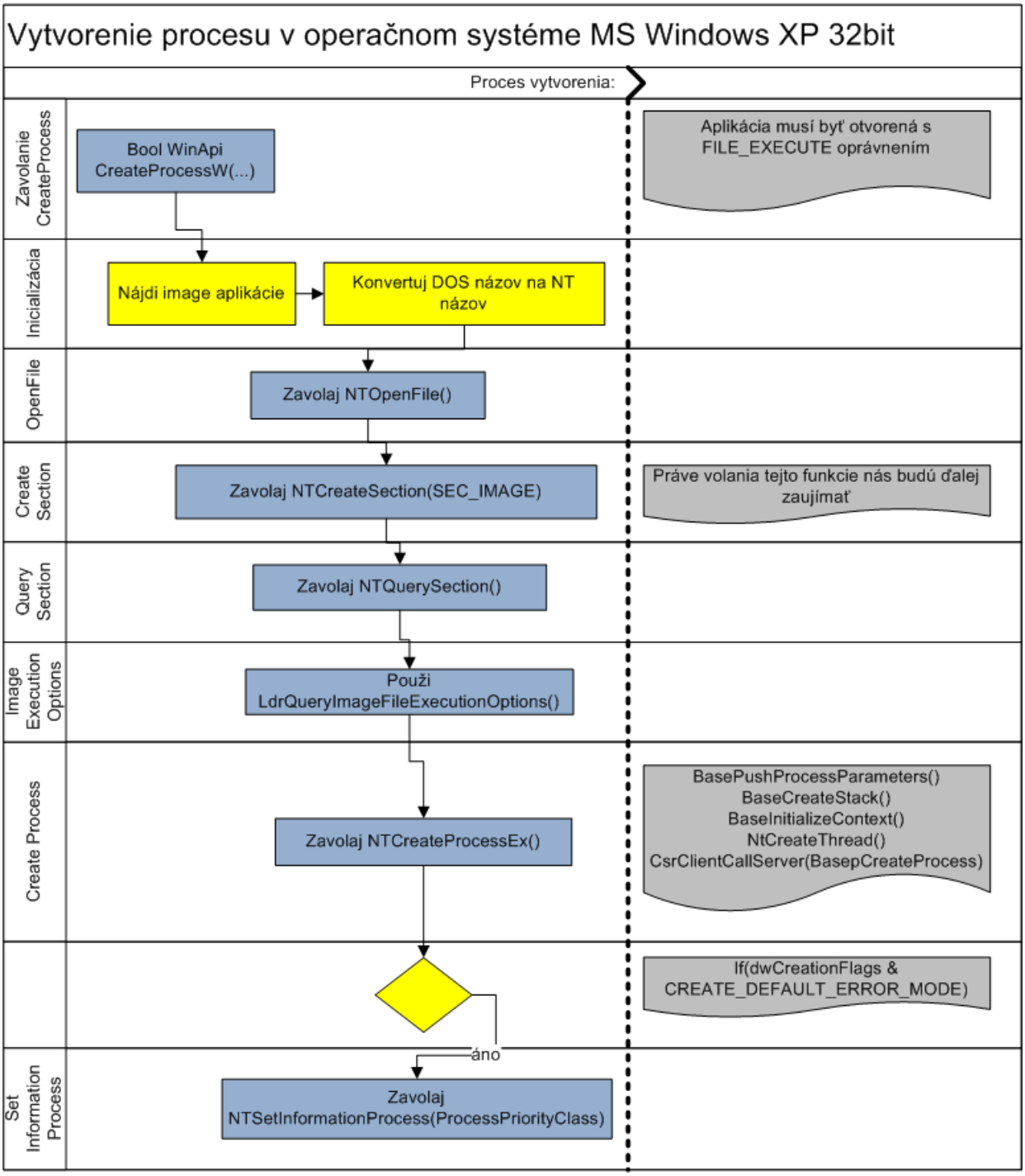
---

[3] Výrazu "všetkými spôsobmi" sa vyhýbame úmyselne; nemožno totiž s istotou tvrdiť, že sú nám známe všetky možnosti spúšťania aplikácií v systéme Windows XP. Aplikácia tiež nepočíta s bezpečnostnými a inými dierami systému, ktoré by umožnili iné spôsoby spustenia aplikácie

[4] Hookovať alebo hookovanie možno preložiť ako hákovať, hákovanie (angličtina); avšak, z dôvodu nezaužívania uvedeného prekladu v slovenčine budeme v ďalšom texte preferovať neprelożené varianty

[5] Samozrejme si nedovolíme tvrdiť, že je nevyhnutnou absolútne. Za nevyhnutnú ju považuje z pohľadu predchádzajúcej poznámky o úplnosti či neúplnosti nášho prehľadu o možnosti spustenia aplikácie v používateľskom režime.

Obr. 1.2: Volanie CreateProcess



1.3 Hookovanie API funkcií a driver

Keďže sme sa už rozhodli vytvoriť driver na úrovni jadra operačného systému Windows a rozhodli sme sa monitorovať a riadiť volanie funkcie NtCreateSection, pozrieme sa na to, ako môžeme prevziať riadenie danej API funkcie. Prevziať kontrolu nad API funkciou znamená detegovať jej volanie, parametre, ale aj rozhodnúť, akú funkcia vráti hodnotu. Takáto kontrola sa nazýva hook, teda funkcia je hooknutá. Častými implementátormi hookov boli, predovšetkým v posledných rokoch, napríklad spoločnosti vytvárajúce antivírusy, firewally či iné ochranné mechanizmy, napríklad zamedzujúce kopírovaniu dátových nosičov. Podľa [4], vytváranie hookov prináša nasledovné 4 hlavné možnosti:

1.	Monitorovanie činnosti API funkcie
2.	Debugovanie a reverzné inžinierstvo
3.	Prenikanie do nezdokumentovaných alebo zle zdokumentovaných častí operačného systému
4.	Rozširovanie funkcionality pôvodných funkcií

Naším cieľom bude uplatniť predovšetkým 4., ale aj 1. bod z vyššie uvedenej tabuľky možností. Budeme sa totiž snažiť monitorovať využitie funkcie NtCreateSection, ale aj rozšíriť jej funkcionalitu povoľovaním spustenia iba pre aplikácie, u ktorých je to žiadané.

Podľa [4], pred implementáciou hooku je dôležité rozhodnúť či potrebujeme riadiť funkcionalitu jednej aplikácie alebo funkcionalitu v rámci celého operačného systému. Tu bolo naše rozhodnutie jednoznačné; potrebujeme monitorovať zvolenú funkciu v celom systéme, a to tak, že adresu našej "proxy" funkcie zapíšeme do **ServiceTable**, teda poľa tabuľky služieb, v objekte **KeServiceDescriptorTable**, do riadku prislúchajúcemu funkcii **NtCreateSection()**; jej adresu v tomto objekte driveru zašleme ako parameter z komunikačnej aplikácie, ktorej popis uvedieme neskôr. Kód driveru v režime jadra bude spúšťaný v subsystéme Native; pre rozšírenie prehľadu uvádzame zoznam subsystémov v systéme Windows XP:

Subsystém	**Použitie**
CONSOLE	Textové aplikácie (konzola)
WINDOWS	Nevyžaduje konzolu, zvyčajne vytvára vlastné okná
NATIVE (/WDM)	Drivery zariadení
POSIX	Aplikácie pre subsystém POSIX
WINDOWSCE	Aplikácie pre zariadenia Windows CE

Tabuľka 1.1: IRQL vo Windows XP

IRQL	**Popis**
PASSIVE_LEVEL	Prerušenia sa nemaskujú Stránkovanie pamäte je dostupné
APC_LEVEL	Maskovanie iba prerušení na úrovni APC Stránkovanie pamäte je dostupné
DISPATCH_LEVEL	Maskovanie prerušení na úrovniach DPC a nižších Stránkovanie pamäte nie je dostupné
DIRQL (Device IRQL)	Maskovanie všetkých prerušení na tejto a nižších úrovniach

[APC] Asynchronous Procedure Call; viac informácií napríklad v [6], s. 366-371

[DPC] Deferred Procedure Call

Podľa [13], iniciačnou funkciou driveru je ľubovoľná funkcia, ktorá sa uvedie ako argument pre linker. Štandardný názov je však **DriverEntry**, pre zvýšenie čitateľnosti kódu sú v navrhovanom systéme podobné konvencie dodržiavané. Keďže driver pracuje v režime jadra, je dôležité rozhodnúť o IRQL[6], s ktorým bude procesor toto vlákno vykonávať. Na základe informácií uvedených v [13], IQRL môže nadobúdať hodnoty uvedené v tabuľke 1.1. Pre naše potreby bude postačujúce IRQL PASSIVE_LEVEL. Nahrávanie do pamäte a deaktivácia driveru sú vykonávané štandardnými procedúrami navrhnutými v [2] a [13]. V prípade potreby je možné zabezpečiť automatické spúšťanie driveru, a to napríklad využitím databázy Registry systému Windows XP; avšak, pri automatickom spúšťaní nie je zabezpečený prístup k externému databázovému serveru, a teda nie je možné uskutočňovať požadované filtrovanie povolených aplikácií. Preto v nami navrhovanom systéme je spúšťanie driveru implementované prostredníctvom komunikačnej aplikácie, ktorá je spúšťaná po inicializácii systému; teda, driver sa spustí práve vtedy, keď sú spustené všetky komponenty, ktoré vyžaduje k svojej korektnej činnosti.

1.4 Komunikácia medzi procesmi

Komunikácia medzi procesmi[7] je pri modulárnych systémoch skladajúcich sa z viacerých procesov nevyhnutnou a dôležitou súčasťou návrhu. Na rýchlosti komunikácie závisí pružnosť systému; v našom prípade je táto rýchlosť esenciálna, keďže hookujeme jednu zo

[6] Interrupt ReQuest Level

[7] Zväčša známa pod skratkou IPC; Inter-Process Communication, preklad (angličtina)

základných API funkcii, a stabilita takejto komunikácie podmieňuje funkčnosť a stabilitu celého navrhovaného systému, ako aj systému operačného. V tejto časti sú naznačené základné parametre, schopnosti a obmedzenia niektorých najpoužívanejších komunikačných prostriedkov využívaných na komunikáciu medzi procesmi v operačnom systéme Windows XP. Zvolené spôsoby komunikácie by tiež mali reflektovať snahu o zjednodušenie prípadnej prenositeľnosti systému na iné platformy.

Memory-Mapped Files

Pre umožnenie komunikácie medzi procesmi si popri vytváraní procesu uveďme dôležité fakty aj o vytvorení MMF[8] v adresnom priestore procesu. Podľa [5], pre vytvorenie MMF je potrebné vykonať nasledovné 3 kroky:

1. Vytvoriť alebo otvoriť súbor ako objekt jadra identifikujúci súbor na disku, ktorý chceme ako MMF použiť. Toto vykonáme prostredníctvom funkcie **CreateFile**, kde možno nastaviť pre oprávnenia hodnoty: 0 pre možnosť iba čítať vlastnosti súboru, GENERIC_READ pre čítanie zo súboru, GENERIC_WRITE pre zápis súboru; hodnoty možno tiež kombinovať, teda pre umožnenie čítania aj zápisu sa použije konštrukcia GENERIC_READ | GENERIC_WRITE; pre zdieľanie možno nastaviť hodnoty: 0, kedy všetky iné pokusy o otvorenie súboru zlyhajú, FILE_SHARE_READ, kedy budú povolené iba pokusy o otvorenie na čítanie, FILE_SHARE_WRITE, kedy budú povolené iba pokusy o otvorenie na zápis, a analogicky skombinovaná hodnota FILE_SHARE_READ — FILE_SHARE_WRITE

2. Vytvoriť objekt jadra mapujúci súbor, ktorý systém informuje o veľkosti súboru a spôsobu prístupu k nemu. Toto vykonáme prostredníctvom funkcie **CreateFileMapping**. Tejto funkcii ako parameter nastavíme ochranu stránkovania pamäte, kde sa povolí buď PAGE_READONLY alebo PAGE_READWRITE, alebo PAGE_WRITECOPY, alebo PAGE_EXECUTE_READ, alebo PAGE_EXECUTE_READWRITE. Pri každej z uvedených možností je potrebné nastaviť potrebné parametre už pri volaní CreateFile.

3. Vyzvať systém k namapovaniu súboru alebo jeho časti do adresného priestoru procesu, čo sa vykoná prostredníctvom funkcie **MapViewOfFile**, pri ktorej volaní uvedieme zamýšľané činnosti so súborom, a to konkrétne hodnotu z množiny obsahujúcej hodnoty FILE_MAP_WRITE, FILE_MAP_READ, FILE_MAP_ALL_ACCESS,

[8]Súbor mapovaný v pamäti (Memory-Mapped File), preklad (angličtina)

FILE_MAP_COPY a FILE_MAP_EXECUTE. Nastavenia súboru a požadované činnosti musia byť kompatibilné.

Presné požiadavky a parametre jednotlivých vyššie uvedených funkcií možno nájsť v [5] v kapitole 17. Podobne, po ukončení využívania MMF, je potrebné vykonať 3 kroky:

1. Požiadať systém o odmapovanie objektu jadra mapujúceho súbor z adresného priestoru procesu, čo sa vykoná prostredníctvom funkcie **UnmapViewOfFile**
2. Zavrieť objekt jadra mapujúci súbor
3. Zavrieť objekt jadra

Táto metóda komunikácie medzi procesmi je dobre použiteľná predovšetkým na komunikáciu medzi aplikáciami naprogramovanými v C a C++, pretože v týchto jazykoch je umožnená jednoduchá správa súborov, ich mapovanie, zapisovanie, atp. Preto MMF využijeme ako jednoduché komunikačné rozhranie medzi dvoma súčasťami nami navrhovaného systému. Za menej využiteľný tento spôsob považujem na účely komunikáciu s aplikáciou naprogramovanou v Jave, keďže na korektné spracovanie pamäťou mapovaného súboru je zrejme potrebné použiť JNI[9]. Podľa [12] takýto program totiž v istom slova zmysle už nemožno ani považovať za aplikáciu napísanú v programovacom jazyku Java. Prišli by sme tím tiež o možnosť jednoduchej prenositeľnosti na platformy UNIX, Linux, Mac OS X a ďalšie, kde sú MMF podporované.

Pri implementácii týchto krokov je dôležité sa zamyslieť nad flat modelom pamäte procesov v uvažovanom operačnom systéme. Z tohto dôvodu nie je totiž na uchovávanie a zdieľanie hodnôt možné použiť globálne premenné alebo iné vzhľadom na proces lokálne štruktúry. Zaujímavým je i fakt, že rôzne pokročilé komunikačné mechanizmy, ktoré ponúka operačný systém Windows, sú založené práve na pamäťou mapovaných súboroch.

Rúry

Anonymné rúry[10] umožňujú[11] jednosmernú[12] medziprocesovú komunikáciu priamočiarym zasielaním bajtov medzi komunikujúcimi procesmi. Teda, na obojstrannú komunikáciu je potrebné použiť rúry dve. Zvyčajne jeden proces disponuje handle na čítanie a druhý

[9] Java Native Interface
[10] Preložené z Anonymous Pipes (angličtina)
[11] Zdroj: [5], s.380-383
[12] Preložené z half-duplex(angličtina)

handle na zápis do rúry. Z dôvodu prakticky úplnej absencie implementácie rúr v programovacom jazyku Java je ich využitie na komunikáciu s Java aplikáciou značne komplikované.

Rozšírením anonymných rúr sú tzv. pomenované rúry[13]. Základne rozdiely medzi pomenovanými a anonymnými rúrami možno zhrnúť do nasledovných bodov[14]:

1. Orientácia na správu - umožňuje rôzne dĺžky správ
2. Obojsmernosť, nie je teda potrebné na vzájomnú komunikáciu používať dve rúry
3. Možnosť existencie viacerých samostatných inštancií rúr s rovnakým názvom
4. Jednoduchá a priamočiara komunikácia v rámci siete
5. Rozšírené možnosti a funkčnosť zjednodušujú request-response a client-server komunikáciu

Absencia implementácie oboch spomenutých variánt rúr v jazyku Java však opäť komplikuje ich využitie na komunikáciu medzi procesmi, ak je jeden z nich v programovacom jazyku Java naprogramovaný.

Mailsloty

Mailsloty sú pomenované mechanizmy na broadcastovú komunikáciu. Ich vlastnosti možno zhrnúť do nasledovných bodov[15]:

1. Jednosmernosť
2. Zvyčajné využitie na one-to-many komunikáciu
3. Klient nie je informovaný o doručení správy
4. Možnosť využitia prostredníctvom siete
5. Dĺžka správ je obmedzená

I keď sú mailsloty nepochybne zaujímavou možnosťou komunikácie, v nami navrhovanom systéme zrejme nebudú mať praktické využitie.

[13]Preložené z Named Pipes (angličtina)
[14]Zdroj: [5], s.384-392
[15]Zdroj: [5], s.401-403

Sockety

Komunikácia medzi procesmi prostredníctvom socketov[16] je štandardnou a široko podporovanou i využívanou metódou výmeny dát medzi aplikáciami nie len v prostredí Windows, ale na väčšine dnes používaných operačných systémov, čo podľa [5] umožnilo odvodenie implementácie v jednotlivých systémoch z konceptu Berkeley Sockets; podľa uvedeného zdroja by mali byť i knižničné funkcie využívané v 64-bitových verziách Windows rovnaké ako tie vo verziách 32-bitových. Jednoduchú implementáciu komunikačného prostredia medzi aplikáciami vytvorenými v programovacích jazykoch Java a C++ možno nájsť napríklad v [7]. Nepochybnou výhodou komunikácie prostredníctvom socketov je jej univerzálnosť; teda, cez socket možno implementovať komunikáciu medzi širokým spektrom aplikácií. I napriek tomuto faktu uvedený spôsob nemožno výhodne použiť na komunikáciu s driverom, ktorý je súčasťou navrhovaného systému. Vzhľadom na využitie komunikácie výlučne na komunikáciu v rámci jedného počítača, táto komunikácia nie je natoľko náchylná na potenciálny útok ako komunikácia medzi počítačmi v sieti; preto podľa [5] nie je nevyhnutnou implementácia SSL[17].

Komunikácia prostredníctvom socketu sa skladá z klienta a serveru. Klient sa pripojí na jemu známy port, na ktorom server počúva prichádzajúce spojenia. Na rozdiel od mnohých iných spôsobov komunikácie takto možno spracovať i viacero pripojení. Spôsoby komunikácie prostredníctvom socketov možno rozdeliť na 2 skupiny:

1. Spoľahlivá stateful komunikácia prostredníctvom protokolu TCP/IP, použije sa parameter **SOCK_STREAM**

2. Nespoľahlivá stateless komunikácia prostredníctvom protokolu UPD, použije sa parameter **SOCK_DGRAM**

Tiež je možné využiť iný protokol; v našom prehľade sa touto možnosťou však nebudeme zaoberať. Podľa [5], spoľahlivá komunikácia zabezpečí doručenie správy na cieľovú stanicu alebo nás o jej nedoručení informuje, čo je, vzhľadom na kritickosť komunikácie v navrhovanom systéme, zrejme vhodná vlastnosť. Nespoľahlivá komunikácia vlastne v svojej podstate ani nie je komunikáciou medzi dvoma entitami a využíva sa napríklad pri RPC[18]. Nutnosť implementovať handshaking alebo iné overovanie komunikácie nad

[16]Socket by bolo možné preložiť ako zásuvka; avšak, z dôvodu zaužívania výrazu Socket aj v našich podmienkach budeme v ďalšom texte preferovať výraz socket

[17]Secure Sockets Layer

[18]Remote Procedure Calls

protokolom UDP by implementáciu v našom prípade zbytočne komplikovala. Z uvedených dôvodov komunikáciu prostredníctvom Socketov s využitím spoľahlivej komunikácie neskôr využijeme na komunikáciu medzi komponentami nami navrhovaného systému.

1.5 Virtualizácia a hypervisoring

Virtualizácia je zaujímavou, rýchlo sa vyvíjajúcou a rozširujúcou metódou, ktorá umožňuje v rámci jedného operačného systému, ktorý tiež možno nazvať hostiteľským, súčasný beh jedného alebo viacerých ďalších operačných systémov. Hardvér je týmto virtuálnym operačným systémom poskytnutý sprostredkovane, a to aj s pomocou špeciálnej podpory virtualizácie na nových procesoroch. I keď je zrejmé, že virtuálny operačný systém má nižšiu výkonnosť ako nevirtuálny operačný systém, jeho využitie má mnohé výhody; z hľadiska bezpečnosti je to predovšetkým možnosť behu potenciálne nebezpečných aplikácií bez reálneho ohrozenia dát či systému mimo virtuálneho operačného systému (samozrejme v prípade, že tomuto virtuálnemu systému nie sú dané dáta sprístupnené).

Hardvérová virtualizácia alebo tiež hypervisoring, predstavený už pred vyše 40 rokmi, umožňuje prostredníctvom VMM[19] súčasný beh viacerých operačných systémov na jednom počítači. Hypervízor monitoruje prácu hosťovských operačných systémov, ktorým je umožnené i zdieľanie hardvérových prostriedkov. Jedným z rizík takéhoto usporiadania je možnosť inštalácie malware ako hypervízora, čo výrazne komplikuje detekciu a odstránenie takýchto aplikácií.

Spomenuté spektrá riešení využívajúce virtualizáciu zažívajú v posledných mesiacoch významnú renesanciu a je veľmi pravdepodobné, že budú vo svete informačných technológií zastávať ďaleko významnejšiu úlohu, ako je tomu dnes. Vzhľadom na dostupnosť a komplikovanosť sa nimi však v tejto práci nebudeme ďalej zaoberať.

[19] Virtual Machine Monitor

Kapitola 2

Existujúce prístupy

Motiváciu a usmernenie pri tvorbe aplikácie poskytli i existujúce riešenia. V nasledujúcich sekciách stručne popíšeme niektoré z nich. Keďže sme v tejto práci navrhovali systém pre operačný systém Microsoft Windows XP, vyhneme sa popisu systémov fungujúcich výlučne v alternatívnych operačných systémoch, prípadne systémov v týchto operačných systémoch priamo vbudovaných.

2.1 Lumension® Endpoint Protection

Spoločnosť Lumension je jednou z lastovičiek, ktorá sa podujala riešiť problém obmedzenia spúšťania aplikácií na počítačoch podobným spôsobom, ako sa o to pokúša nami navrhovaný systém. Vyvinula uzavretý systém, ktorý umožňuje používať whitelist na selekciu povolených aplikácií pre počítače predovšetkým vo firmách. K tomuto riešeniu časom pridala i vlastný antivírus s jadrom Norman a ďalšie prvky zabezpečenia.

K nepochybným výhodám tohto riešenia patrí možnosť nasadenia v korporáciách. Tiež all-in-one riešenie môže byť istou skupinou zákazníkov preferované. Veľkou výhodou tohto systému je nasadenie whitelistu ako vhodnej alternatívy dovtedy štandardne využívaných blacklistov; nie je totiž reálne možné analyzovať všetky nebezpečné či nechcené aplikácie a aktualizovať takýmito dátami lokálne databázy na koncových staniciach. Zaujímavou je i široká ponuka koncových klientov pre agentský modul systému, a to od Windows 2000 po Windows 7. K výhodám riešenia patrí aj jeho dlhšia existencia na trhu, ktorá zvyšuje dôveryhodnosť produktu a prispela tiež k zvýšeniu stability a spoľahlivosti.

Uzavretosť systému však spôsobuje značné problémy, ak riešenie nie je plne kompatibilné s politikou klienta. Nemožnosť vykonať alternatívne filtrovanie aplikácií, zmenu

spôsobu detekcie a iných parametrov je obmedzujúca a zväzuje ruky. Tiež spôsob nastavovania povolených aplikácií, ktorý je viazaný na doménu, obmedzuje využitie v inak organizovaných či menších spoločnostiach. Za problém tiež považujem integráciu antivírusového jadra Norman, ktoré rozhodne podľa štatistík nepatrí dlhodobo k špičke na trhu[1], a cenu približne 30 eur za nasadenie na 1 koncovej stanici. Na internete tiež možno nájsť informácie o istých problémoch so stabilitou produktu, ktorým sa však podobný systém zrejme nevyhne a obmedzená podpora 12 jazykov dnes nie je už zďaleka štandardom. Zaujímavým obmedzením je i nutnosť inštalovať produkt na server s nainštalovaným Windows Server 2003 alebo 2008 v anglickej edícii, kde musí byť nastavená angličtina ako predvolený jazyk.

2.2 Application Launch Control v HIPS

Mnohé dnešné zabezpečovacie systémy, či už firewally alebo celé Internet Security balíky, obsahujú HIPS[2] systémy, ktoré obsahujú Application Launch Control systémy, teda produkty spravujúce spúšťanie aplikácií. Mnohé z týchto systémov sú implementované pomocou driveru pracujúceho v režime jadra[3] a využívajú okrem blacklistov aj heuristické a virtualizačné metódy.

Heuristické metódy sú zaujímavým rozšírením, avšak pri využití whitelistov im neprikladám vysokú dôležitosť. Tiež driver v režime jadra je vhodnou voľbou (viď 1.1). Pre domáceho používateľa sú tieto produkty zväčša zaujímavé práve pre svoju jednoduchú použiteľnosť a nízku cenu, ktorá je umožnená okrem iného aj absenciou centralizovaných správcovských komponentov.

Nemožnosť riadenia pre sieť počítačov či centralizovaného uloženia predvolieb však výrazne obmedzuje využitie pre firemné účely či účely iných väčších organizácií. Tento prístup sa ukazuje problematickým aj v domácom alebo inom nasadení v menšom meradle; takéto produkty totiž často využívajú možnosť aktívnej spolupráce s používateľom, ktorý zväčša nerozumie hrozbe, ktorú komunikované položky môžu predstavovať, a teda sa často rozhodne nesprávne, čím je efektivita a zabezpečovacia funkcia systému značne

[1] Štatistikami sa tu myslia testy typu Virus Bulletin, AV-Comparatives.org či Virus.gr; výsledky možno nájsť na internetových stránkach jednotlivých testov

[2] Host-based Intrusion Prevention System; všeobecný význam možno nájsť napr. na Wikipédii; z dôvodu zrozumiteľnosti sa nepokúšame o preklad pojmu

[3] napríklad Sunbelt Personal Firewall, http://www.sunbeltsoftware.com/home-home-office/sunbelt-personal-firewall/

znížená. V prípade reštriktívnejšieho prístupu predvoleného nepovolenia akcie pre zmenu hrozí nastavenie príliš veľkých výnimiek používateľom alebo nevôľa používateľa takýto systém používať. Tiež voľba blacklistu pri voľbe politiky predvoleného povolenia spustenia aplikácie je síce tradičným, ale menej úplným a spoľahlivým riešením.

2.3 Microsoft Vista blacklist

Po dlho očakávanom vydaní operačného systému Microsoft Windows Vista, jednou z nových súčastí nového systému bol aj blacklist, ktorý dohliada na problémy s kompatibilitou aplikácií. Pri pokuse o spustenie aplikácie systém overí či sa názov aplikácie nenachádza v blackliste, ktorý je lokálne uložený na počítači a je súčasťou systému.

K výhodám tohto systému patrí, že existuje. Teda, že sa výrobca pokúša istým spôsobom chrániť používateľa pred potenciálnymi rizikami plynúcimi zo spustenia nekompatibilnej aplikácie. Tiež voľba blacklistu je tu zrejme nevyhnutnosťou, pretože nie je reálne možné kumulovať názvy všetkých aplikácií, ktoré sú s daným operačným systémom kompatibilné. Za dobré riešenie možno považovať aj sťahovanie aktualizácií tohto blacklistu z Microsoft Update, čo je pre tento operačný systém štandardným postupom.

Naopak, k nevýhodám patrí, že[4] tento blacklist filtruje aplikácie na základe ich názvu; teda ľubovoľná zmena názvu alebo, naopak, neúmyselná voľba obsiahnutého názvu pre kompatibilnú aplikáciu môže mať nečakané a predovšetkým nechcené dôsledky.

[4]na základe osobných skúseností, bohužiaľ bližšia dokumentácia systému nebola nájdená

Kapitola 3

Ukážka implementácie riešenia

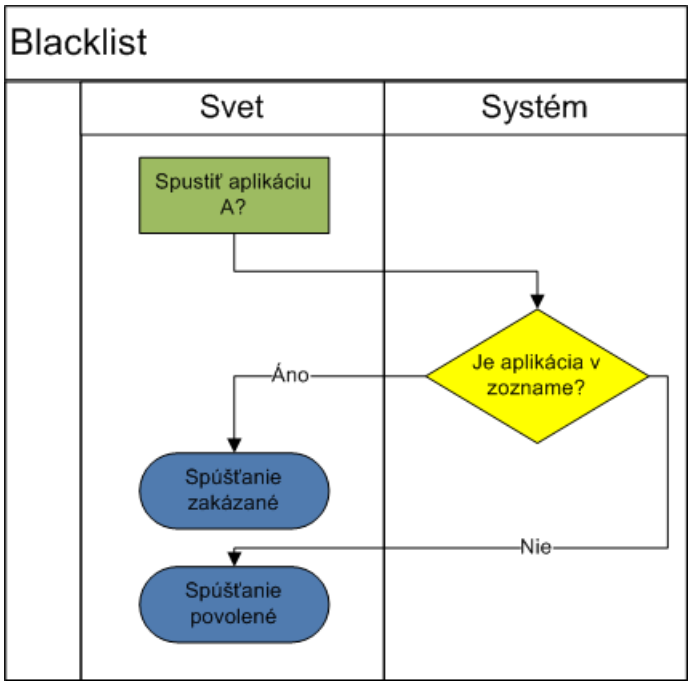
V tejto kapitole bude načrtnutá jedna z možností implementácie riešenia uvedeného problému v prostredí MS Windows XP SP3. Tiež sa budeme venovať ďalším možnostiam vývoja našej aplikácie a jej ďalšiemu využitiu.

3.1 Blacklisty a whitelisty

Jedným zo základných rozhodnutí, ktoré má rozhodujúci vplyv na ďalší návrh aplikácie, je uvážiť, či použiť riešenie formou blacklistu alebo whitelistu. Zatiaľ čo riešenie pomocou blacklistu umožňuje používateľom systému širší pohyb, nebráni spúšťaniu potenciálne dobrých aplikácií, prípadne ich nových verzií, využitie whitelistu je viac reštriktívne. Používateľ môže spúšťať iba dané aplikácie, v danej verzii, v našej implementácii dokonca iba s daným názvom, bez prakticky akejkoľvek možnosti zmeny (toto zabezpečujú hašovacie funkcie, ktoré sú využívané na overenie identity aplikácie; o ich implementácii si viac povieme v časti 3.2.1).

Z uvedených dôvodov je dôležité hlbšie analyzovať cieľovú skupinu navrhovaného systému. Vo väčších spoločnostiach, kde by mali byť počítače využívané prevažne na presne definované ciele, a to na základe špecifickej politiky klienta a dlhodobo testovanými aplikáciami (testujú sa zvyčajne konkrétne verzie pre zamedzenie prípadným konfliktom), je reštriktívne riešenie žiadúce. Oddelenie správy počítačov totiž už nebude musieť riešiť problémy vzniknuté z iniciatívnymi používateľmi nainštalovaných aktualizácií a nekompatibilných nových verzií aplikácií kolidujúcich s inými dôležitými systémami. Tiež sa nám podarí odbúrať časť útokov vedených prostredníctvom sociálneho inžinierstva, kde sa aplikácie vydávajú za iné, žiaduce, prípadne i schválené aplikácie; túto možnosť značne

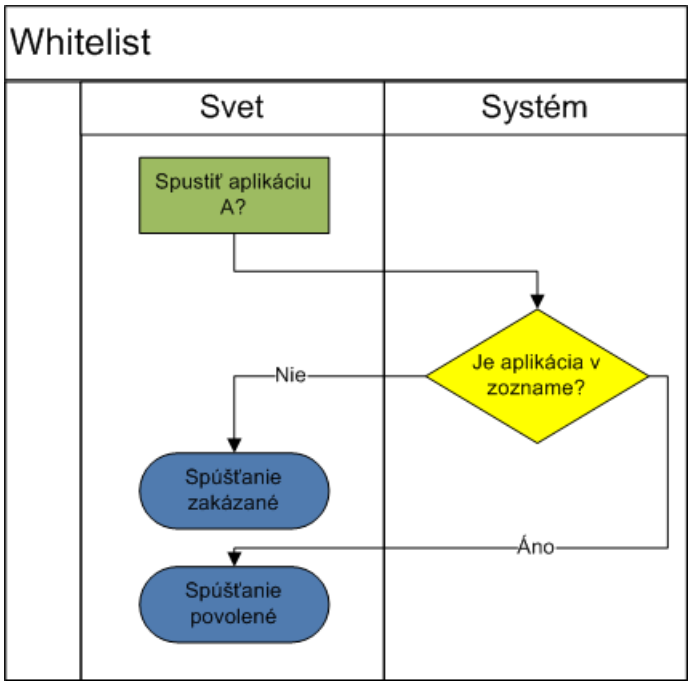
Obr. 3.1: Model fungovania blacklistu



(prakticky takmer úplne) redukuje využitie už spomenutých hašovacích funkcií. Pri použití blacklistu by bolo napríklad možné povoliť spustenie aplikácie aj zmenou jej názvu. Ak by sme blacklist obmedzili na haše aplikácií, stačilo by aplikáciu prakticky akokoľvek upraviť, teda pre jej zakázanie by bolo nutné pracne vyhľadávať všetky dostupné verzie danej aplikácie. V prípade doplnenia blacklistu pokročilými heuristickými metódami by systém nie len vyžadoval viac procesorového času, ale aj pripúšťal možnosť falošných (neodôvodnených) povolení alebo zakázaní spúšťania aplikácií. Aj preto sme sa rozhodli v implementácii navrhovaného riešenia využiť princíp whitelistov. Porovnanie princípu funkcie blacklistu a whitelistu možno sledovať na obrázkoch 3.1 a 3.2.

Avšak, dôležité je poznamenať, že vďaka vysokej modularite systému je zmena prístupu otázkou veľmi jednoduchého zásahu. Teda, ak by pre konkrétne potreby daného klienta bolo potrebné implementovať navrhovaný systém prostredníctvom blacklistu, išlo by o relatívne lacnú zmenu z pohľadu práce na zmene; tiež implementácia kombinovaného riešenia, kde by sa politika mohla meniť od počítača k počítaču, by nebola komplikovaná, a to vďaka flexibilného návrhu GUI komponentu bližšie opísaného v 3.2.2.

Obr. 3.2: Model fungovania whitelistu



3.2 Štruktúra riešenia

Na základe analýzy problému, ktorú sme rozobrali v predchádzajúcich kapitolách, možno navrhnúť štruktúru riešenia, ktoré by sa vyhlo chybám a problémom, ktoré sa nachádzajú v existujúcich riešeniach podobných problémov. Systém by zrejme mali používať 3 skupiny používateľov:

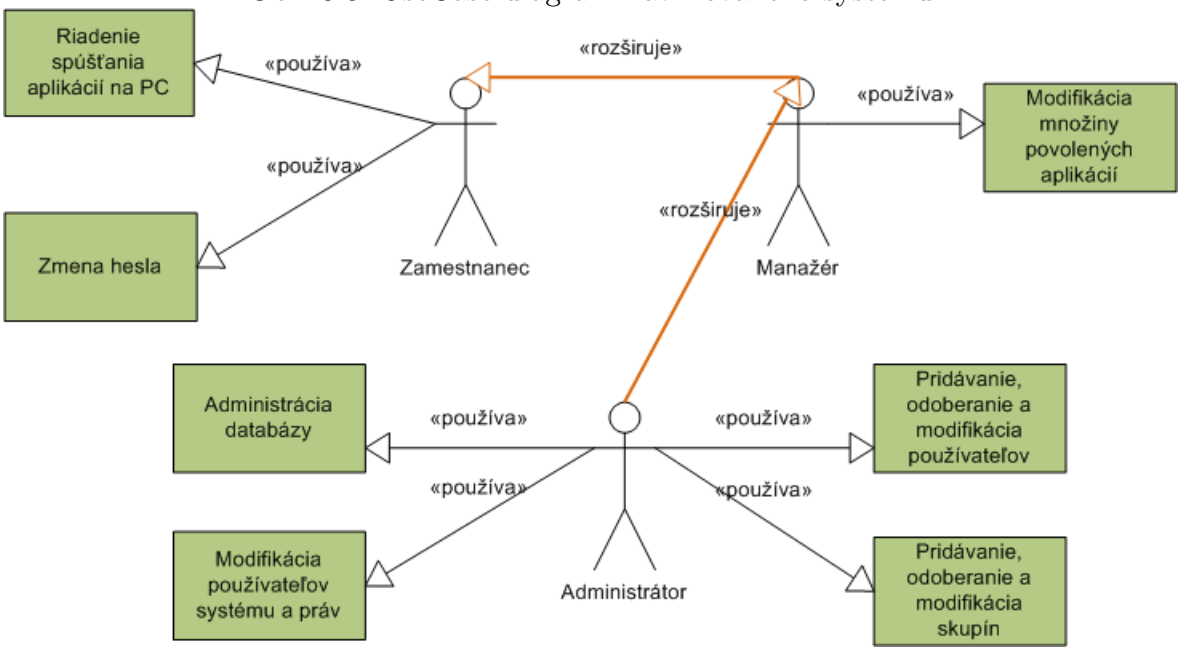
1. **Bežný používateľ** (zamestnanec), ktorý iba využíva funkčnosť systému na riadenie spúšťania aplikácií na jeho počítači, a prípadne si vie zmeniť svoje heslo

2. Ďalšou skupinou používateľov budú **manažéri** (riaditelia, správcovia skupín), ktorí budú okrem bežného používania systému aj pridávať, odstraňovať či modifikovať množinu povolených aplikácií

3. Poslednou skupinou budú **administrátori** (správcovia systému), ktorí okrem možnosti využívať funkčnosť navrhnutú pre predchádzajúce 2 skupiny používateľov budú mať priamy prístup do databázy pre prípadnú možnosť pridávania ďalších administrátorov, ako aj práva pomocou navrhovaného systému pridávať, odstraňovať a modifikovať používateľov a používateľské skupiny

Využívanú funkčnosť, rozdelenie právomocí, ako aj dedenie medzi skupinami používateľov sme znázornili v UML UseCase diagrame 3.3.

Obr. 3.3: UseCase diagram navrhovaného systému

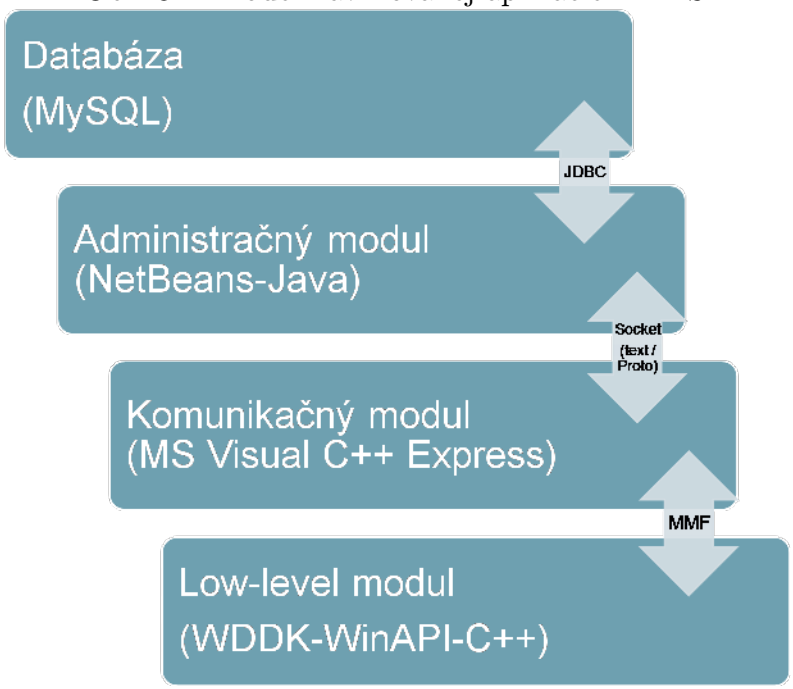


Vychádzajúc z potreby vysokej modularity systému, pokúsili sme sa vytvoriť pre každú súčasť samostatný modul, ktorý by bol v prípade potreby nahraditeľný iným modulom bez nutnosti zmien v ostatných komponentoch. Implementácia vypracovaná ako súčasť tejto práce má nasledovné súčasti:

Komponent	**Typ**	**Jazyk**	**Popis**
AMBS_DB	Databáza	MySQL	Správa práv používateľov Správa whitelistu aplikácií
AMBS_AA	GUI	Java (NetBeans)	Nastavenie aplikácie na pracovnej stanici Komunikácia s používateľom Analýza aplikácií Správa logov
AMBSLL_ DCom	Aplikácia	C++ (MS Visual C++ Express)	Komunikácia medzi driverom a GUI Primárne spracovanie výstupu driveru
AMBS_LL	Driver	C++, Assembler (WDDK)	Nízkoúrovňový komponent Hookovanie API Detekcia pokusov o spustenie

Uvedený model spolu s použitými rozhraniami je vizualizovaný na obrázku 3.4. Dôkladný popis komponentov a komunikačných rozhraní bude uvedený ďalej.

Obr. 3.4: Model navrhovanej aplikácie AMBS



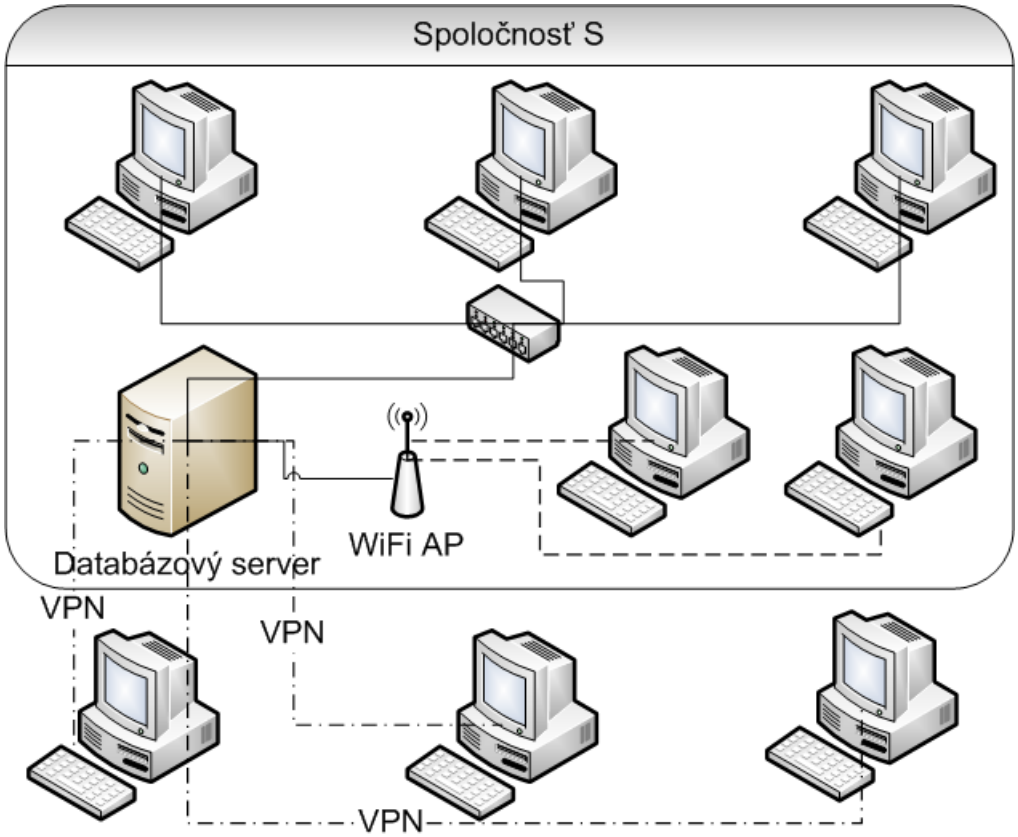
Na základe uvedenej štruktúry možno vytvoriť model priebehu vytvorenia požiadavky na spustenie aplikácie. Ako vstup môžeme uvažovať zavolanie systémovej API funkcie NtCreateSection; v prípade, že by sa nízkoúrovňový modul, driver, v budúcnosti vymenil za iný modul, ktorý by prípadne monitoroval iné či viac API funkcií, a zvyšok návrhu by ostal s naším návrhom kompatibilný, stačilo by v modeli priebehu zmeniť iba tento vstupný parameter. Diagram modelu priebehu je zobrazený v 3.6. Uveďme tiež príklad fyzickej organizácie systému v spoločnosti S; systém by sa zrejme skladal z nasledujúcich základných komponentov:

Databázový server
Sieťové komponenty
(Virtuálna) lokálna sieť
Koncové používateľské stanice pripojené na lokálnu sieť
Koncové používateľské stanice pripojené napr. prostredníctvom VPN

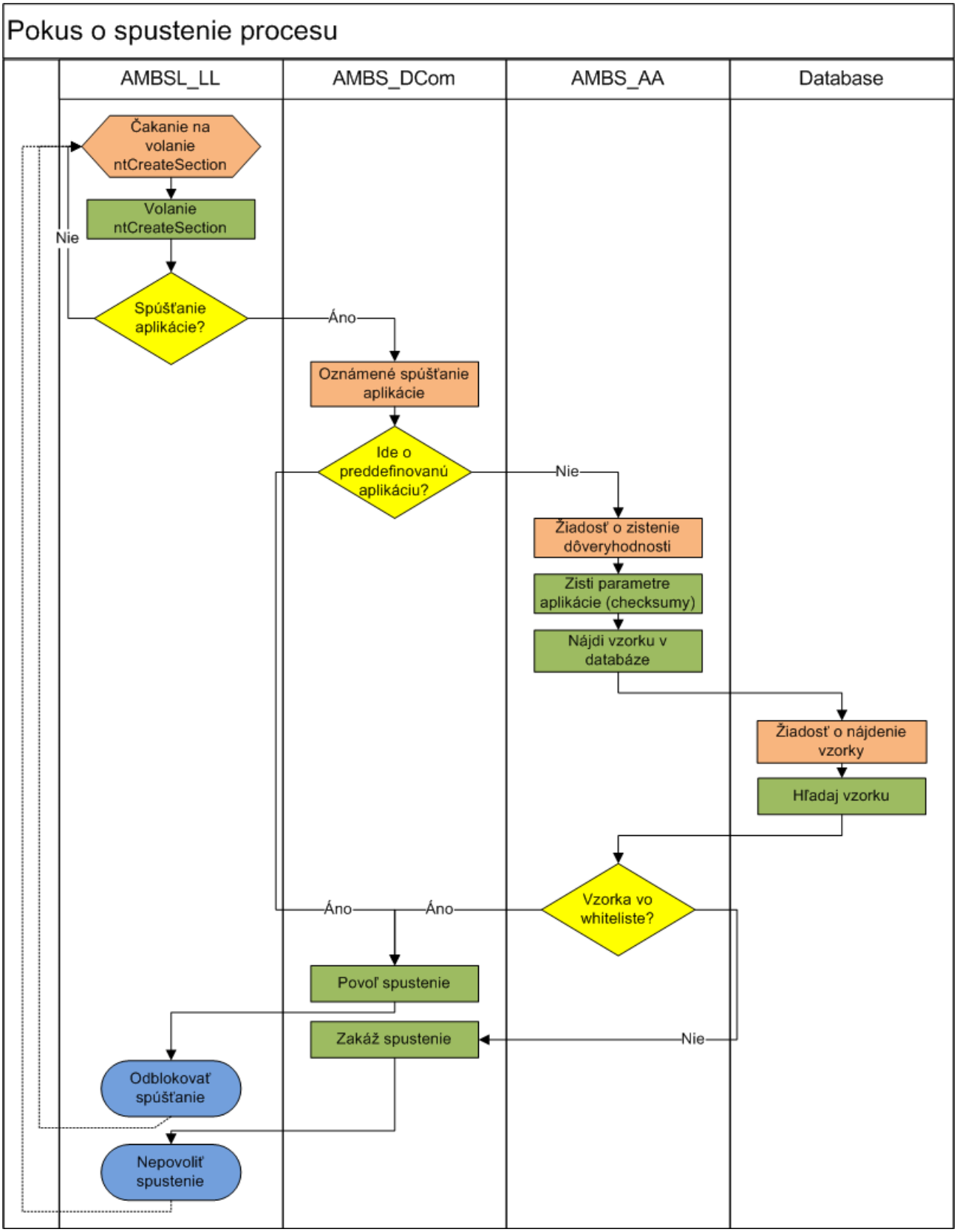
Všeobecná schéma fyzickej organizácie navrhovaného systému s popisom jednotlivých komponentov je uvedená v diagrame 3.5.

Ako vidno z diagramu aktivít 3.6, logicky organizovaný a transparentný beh programu umožňuje veľkú mieru nezávislosti jednotlivých komponentov, a teda široké možnosti rozšíriteľnosti do budúcnosti. Práve jednoznačnosť úloh a vyžadovanej funkčnosti pre jednotlivé

Obr. 3.5: Fyzická organizácia navrhovaného systému AMBS



Obr. 3.6: Diagram modelu priebehu pre aplikáciu AMBS



komponenty navrhovaného systému umožňuje vymeniť komponent za iný, ktorý dodržiava zavedené komunikačné rozhrania, a to bez akéhokoľvek priameho vplyvu na ostatné súčasti systému. V nasledujúcich riadkoch bližšie objasníme fungovanie jednotlivých komponentov.

3.2.1 Databáza

Databáza je jedinou súčasťou navrhovaného systému, ktorá nie je lokalizovaná priamo na koncovom počítači. Z dôvodu snahy umožniť jej centralizáciu, a tým ľahšiu a lepšie synchronizovateľnú modifikáciu, je umiestnená na servery a dostupná napríklad prostredníctvom lokálnej siete alebo sietí VPN.

Keďže jediným využívaným rozhraním na komunikáciu s databázou je JDBC, ktoré je známe svojou univerzálnosťou a širokou kompatibilitou, konkrétna distribúcia databázy je závislá skôr od konkrétnej bezpečnostnej politiky používateľa ako od požiadaviek navrhovaného systému. Z dôvodu využívania niektorých pokročilých funkcií je však nevyhnutné, aby databáza spĺňala nasledujúce požiadavky a aby aspoň jeden z používateľov navrhovaného systému mal oprávnenia na uvedené akcie:

Požiadavka	**Nutnosť**
Vytvorenie databázy (Create database)	Odporúčané
Odstránenie databázy (Drop database)	Odporúčané
Vytvorenie tabuľky (Create table)	Nevyhnutné
Odstránenie tabuľky (Drop table)	Nevyhnutné
Nastav. práv pre tabuľky (Grant)	Nevyhnutné
Nastav. práv pre riadky (Grant)	Vhodné
Nastav. úrovne práv (Grant Update, Select, Delete ...)	Nevyhnutné
Skupiny používateľov	Vhodné

Keďže nebola v čase návrhu aplikácie nájdená bezplatná distribúcia stabilnej databázy

podporujúcej možnosť nastavovania práv pre jednotlivé riadky databázy, prispôsobil sa tomu aj návrh databázy. Zo spektra ponúkaných bezplatných riešení sme siahli po databáze MySQL. V nasledujúcej tabuľke si popíšme základné funkčné požiadavky kladené na databázu:

Požiadavka	**Implementácia**
Ukladať vzorky schválených aplikácií	Áno
Ukladať a manažovať používateľov	Áno
Ukladať a manažovať skupiny používateľov	Áno
Viacúrovňové oprávnenia	
:: iba čítať schválené položky	Áno
:: Pridávať nové aplikácie v manažovaných skupinách	Áno
:: Manažovať používateľov a skupiny	Áno

Na základe požiadaviek na normalizáciu a usmernení uvedených v [9] bola navrhnutá štruktúra zobrazená na obrázku 3.7.

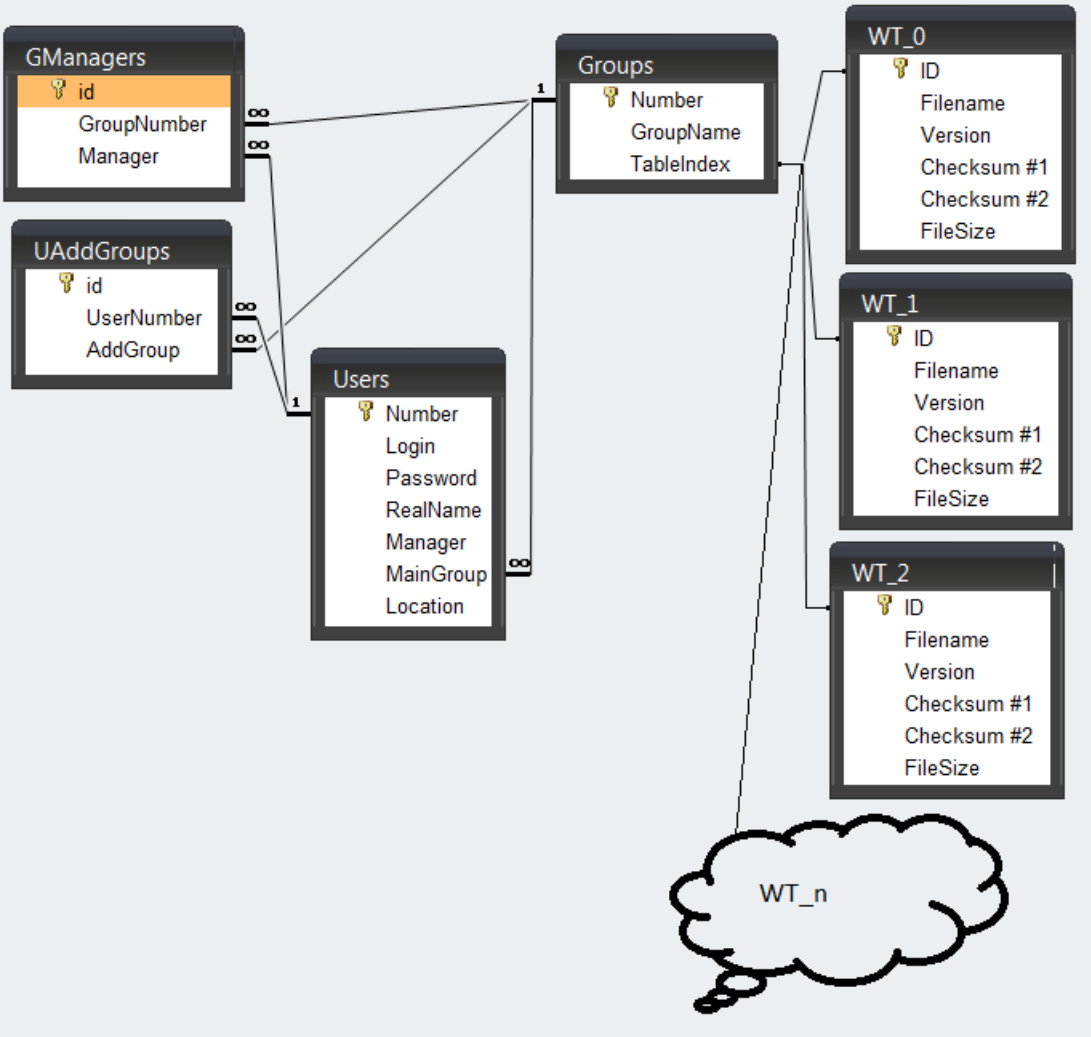
Tabuľka **Users** obsahuje dáta o používateľoch systému. Položka *Number* je primárnym kľúčom - teda číslo používateľa v systéme bude jedinečné. *Login* používateľa je tiež jedinečný, pretože je previazaný na prihlasovanie do databázy, kde jedinečnosť loginu vyžaduje implementácia zvolenej databázy. Položka *Password* sa nevyužíva, pretože heslo používateľa sa ukladá v štandardnej podobe v predvolenej tabuľke zvolenej implementácie databázy. *RealName* predstavuje skutočné meno používateľa. *Manager* odkazuje na číslo používateľa, ktorý je priamy nadriadený daného používateľa[1]. *MainGroup* predstavuje hlavnú alebo predvolenú skupinu používateľa. Položka *Location* je nepovinná; obsahuje IP adresu, z ktorej sa používateľ do databázy prihlasuje; predvolená je hodnota ľubovoľná lokácia (v zvolenej implementácii databázy je to hodnota '%').

Tabuľka **Groups** predstavuje abstrakciu skupín používateľov. Bohužiaľ, použitá databáza ich priamo neimplementuje, takže ich je potrebné simulovať. Skupiny predstavujú aj jednotlivé oddelenia v spoločnosti, aj možnosť zaviesť skupiny globálne povolených aplikácií, napríklad súčastí používaných operačných systémov. Takéto skupiny stačí používateľom priradiť ako dodatočnú skupinu, a tak netreba dané aplikácie povoľovať pre každé oddelenie zvlášť. Skupina má primárny kľúč - svoje číslo (*Number*), názov (*GroupName*) a obsahuje referenciu na tabuľku s povolenými aplikáciami (*TableIndex*).

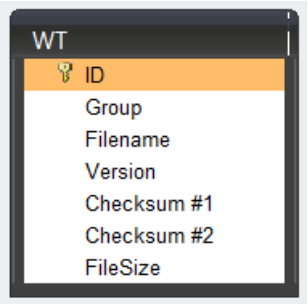
Tabuľka **GManagers** ukladá manažérov skupín, teda používateľov, ktorí majú oprá-

[1]Ide o reláciu na firemné pomery; položka nie je esenciálna pre fungovanie systému

Obr. 3.7: Návrh databázy



Obr. 3.8: Optimalizácia databázy



nenie pre danú skupinu pridávať, odoberať a modifikovať povolené aplikácie. Okrem identifikačného primárneho kľúča (*id*) obsahuje referenciu na používateľa (*Manager*) a referenciu na skupinu (*GroupNumber*).

Tabuľka **UAddGroups** reprezentuje dodatočné skupiny priradené používateľom. Okrem identifikačného primárneho kľúča (*id*) obsahuje referenciu na používateľa (*UserNumber*) a na skupinu (*AddGroup*), ktorá mu je priradená ako dodatočná.

Tabuľky **WT_0** až **WT_n** obsahujú povolené aplikácie pre dané skupiny. Tabuľka Groups na ne referencuje priamo, ich názvom. Toto rozhodnutie bolo prijaté nie z dôvodov návrhových, ale pre obmedzenia vyplývajúce zo zvolenej implementácie databázy; ak by totiž bolo možné nastavovať oprávnenie samostatne pre každý riadok, tieto tabuľky by boli zlúčené do jednej a miesto uvedených tabuliek by bola v štruktúre databázy tabuľka zobrazená na obrázku 3.8; tu by bola položka *Group* v relácií (referenciou na) s id skupiny, v ktorej je daná položka povolená, prípadne by existovala ďalšia tabuľka, ktorá by optimalizovala databázu pre viacnásobné použitie danej položky (čo však zväčša nie je potrebné vzhľadom na možnosť vytvárať tzv. všeobecné skupiny, ktoré sme popísali vyššie). Každá z tabuliek WT_x obsahuje identifikačný primárny kľúč *ID*, názov súboru aplikácie *Filename*, 2 kontrolné haše *Checksum #1* a *Checksum #2*, veľkosť súboru *FileSize* a rezervovanú položku *Version* pre neskoršie možné využitie. Ako kontrolné haše sú použité funkcie MD5 a CRC32; v prípade potreby je však vďaka modularite navrhovaného systému možné tieto funkcie ľahko zmeniť za iné; takáto zmena by znamenala iba nutnosť prepočítať haše pre aplikácie, ktoré už sú v databáze.

3.2.2 GUI

Ako bolo uvedené už v prehľade komponentov, GUI pomenované AMBS_AA[2] je aplikáciou s grafickým používateľským rozhraním naprogramovanou v Jave v prostredí NetBeans. Používateľské rozhranie je vybudované využitím štandardných Swing komponentov a jeho funkcionalita, organizácia a vzhľad sú motivované návrhmi a pripomienkami mnohých používateľov počítačov, ako aj pozorovaním používateľských rozhraní podobných aplikácií už na trhu dostupných.

Táto aplikácia priamo komunikuje s databázou prostredníctvom JDBC, využitie ktorého bolo aj dôležitým faktorom pri rozhodovaní sa o využití programovacom jazyka Java. Dostupné pomerne jednoducho použiteľné štandardné rozhrania s dobrou technickou podporou a pružným vývojom totiž dopĺňa ešte ODBC, ktoré by sa dalo využiť napr. v aplikácií naprogramovanej v C++ či .NET. Avšak na základe analýzy týchto dvoch rozhraní uvedenej v [9] a tu uvedených známych problémov so správou pamäte a stabilitou riešení využívajúcich ODBC, JDBC pôsobí ako jednoduchší prístup.

Okrem JDBC aplikácia disponuje rozhraním pre komunikáciu prostredníctvom socketov; aplikácia implementuje server, na ktorom poskytuje služby porovnania daného súboru so vzorkami v databáze schválenými aktuálnemu používateľovi. Pre implementáciu komunikácie klient-server bola použitá obľúbená implementácia uvedená v [7]. Tieto služby možno poskytovať v štandardnom textovom formáte, ale aj napríklad vo formáte Google ProtoBuf, čo je binárny štandardný formát pre odovzdávanie správ. Oproti XML disponuje napríklad menšou veľkosťou správ či jednoduchším programátorským spracovávaním a keďže jeho zdrojové kódy sú dostupné aj pre Javu, aj pre C++, je pre navrhovanú aplikáciu zaujímavou možnosťou.

Aplikácia AMBS_AA slúži všetkým používateľom, bez ohľadu na úroveň ich privilégií. Každému používateľovi sú prostredníctvom konfiguračného súboru vo formáte INI[3] nastavené prístupové údaje do databázy a niektoré dôležité parametre pre JDBC. Tieto parametre, podobne ako inštaláciu celého produktu, je možné automatizovať, čo je výhodou pre masové nasadenie a administrovateľnosť vo veľkých spoločnostiach. Používatelia s právami manažérov skupín tu môžu pridávať, odstraňovať a modifikovať povolené aplikácie, užívatelia so správcovskými oprávneniami môžu pridávať, odstraňovať a modifikovať i používateľov a skupiny. Vzhľadom na snahu čo najviac zjednodušiť prácu s povolenými

[2] AA ako skratka pre Application Analyzer

[3] V skutočnosti ide o formát Property implementovaný v NetBeans, avšak tento formát je s INI súbormi plne kompatibilný

aplikáciami je možné ich zoznamy importovať i exportovať jednak z iných skupín priamo z databázy, ale aj prostredníctvom CSV[4] súborov.

Ako vyplýva z povahy aplikácie, pre korektné fungovanie systému musí byť spustená - inak by totiž nebolo možné overiť, či je spúšťaná aplikácia povolená. Rozhodovací mechanizmus o povoľovaní aplikácie využíva whitelisty implementované v databáze a rozhoduje na základe pravidiel popísaných v diagrame 3.9. Zaujímavou možnosťou je aj implementácia tzv. "učiaceho módu", ktorý administrátorom dovoľuje sledovať spúšťané aplikácie pri danom biznis procese, a teda automatizovane generovať zoznam aplikácií, ktoré je potrebné danej skupine povoliť. Prostredníctvom aplikácie AMBS_AA možno okrem uvedených činností i sledovať stav celého systému, a to funkčnosť komunikácie prostredníctvom socketov a stav pripojenia k databáze. Štruktúra aplikácie je orientovaná na modularizáciu a čo najväčšiu nezávislosť jednotlivých komponentov z dôvodu uľahčenia ďalších zmien. Jednoduché riešenie rôznych jazykových mutácií prostredníctvom Java Properties tiež umožňuje rýchlu a jednoduchú lokalizáciu aplikácie a voľbu jazyka na základe jazyka preferovaného operačným systémom.

Už spomenutá možnosť lokalizácie, ako aj prehľadná štruktúra používateľského rozhrania a automatizácia mnohých často využívaných alebo komplikovaných činností, ako je počiatočné nastavenie databázy, vytvorenie skupín, používateľov či ich organizácia a manažovanie, modifikácia tabuliek s povolenými aplikáciami, import a export povolení, prispievajú k príjemnej práci s aplikáciou a približujú navrhovaný systém každému používateľovi, ako aj administrátorom celého systému. Z praktických dôvodov aplikácia neobsahuje možnosť vytvoriť nového administrátora aplikácie, ktorá musí byť vykonaná priamo modifikáciou databázy. Aplikácia tiež ošetruje množstvo výnimiek a potenciálnych chýb, a to spôsobom, ktorý používateľovi naznačí, koho a ako má kontaktovať, prípadne administrátorovi pomôže s hľadaním riešenia pre vzniknutý problém.

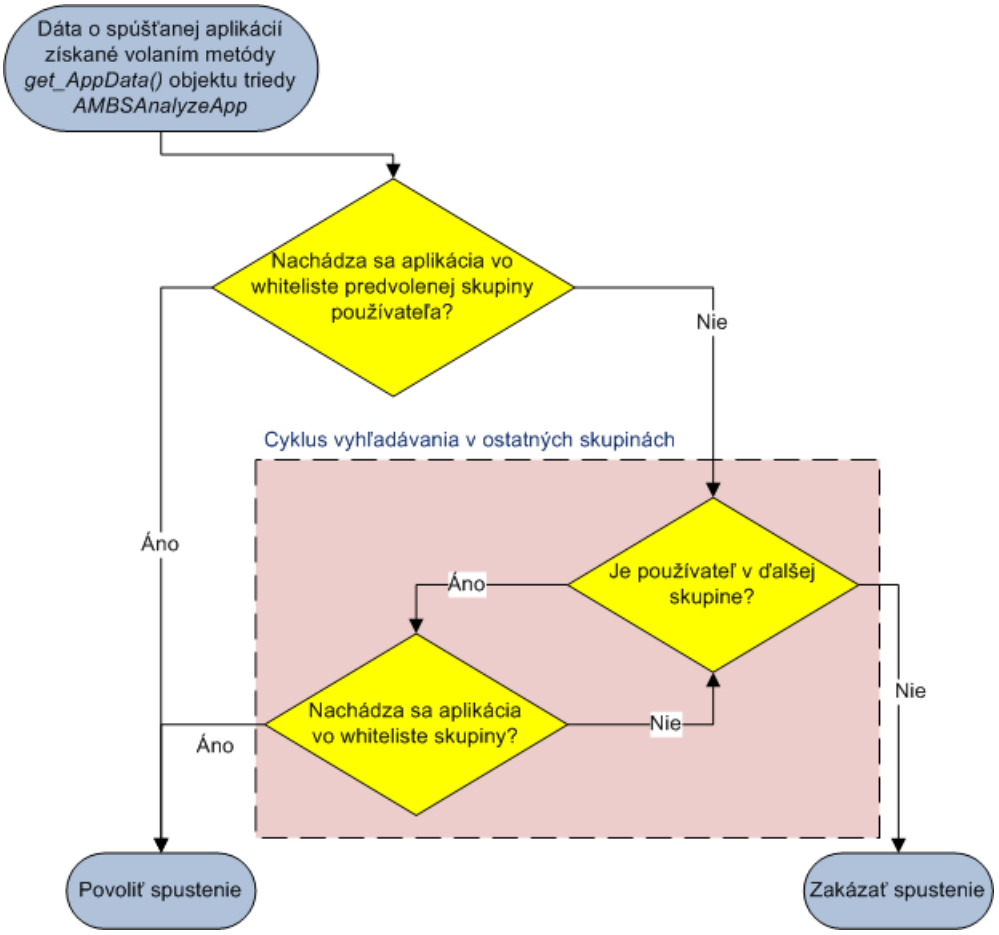
3.2.3 Komunikačná aplikácia

Komunikačná aplikácia AMBSLL_DCom[5] je akousi medzivrstvou, ktorá umožňuje efektívnu komunikáciu medzi AMBS_AA a nízkoúrovňovým driverom. Potreba tejto aplikácie vyplýva z behu driveru v režime jadra; súvislosti sú opísané v 1.1. Z dôvodu dobrej

[4]Comma Separated Values; hodnoty oddelené čiarkou; preklad (angličtina). V praxi sa však tento názov používa aj pre súbory s hodnotami oddelenými iným znakom; v našom prípade je oddeľovačom znak |

[5]LL ako skratka pre Low Level, DCom ako skratka pre Data Communication

Obr. 3.9: Rozhodovanie o povolení alebo zakázaní spustenia aplikácie



kompatibility aplikácií vytvorených prostredníctvom Microsoft Visual C++ Express medzi jednotlivými platformami Microsoft Windows vďaka využitia .NET Frameworku bola zvolená práve táto implementácia; v prípade potreby rozšírenia na iné platformy by malo stačiť aplikáciu prekompilovať v inom kompilátore, prípadne mierne upraviť implementácie niektorých na systém viazaných funkcií. Aplikácia spúšťa a nahráva do pamäte driver, prostredníctvom súboru mapovaného v pamäti mu posiela potrebné parametre, ale i prijíma informácie o volaní systémovej API funkcie NtCreateSection a zasiela odpoveď systému AMBS na toto volanie. Aplikácia prostredníctvom rozhrania opísaného v 3.2.2 vytvára klientské pripojenie na server AMBS_AA a zasiela požiadavku na overenie, či je daná aplikácia pre daného používateľa povolená, alebo nie. Cez toto rozhranie aj prijíma odpoveď.

Pre prípad nefunkčnosti serveru by aplikácia mohla v ďalších rozšíreniach disponovať istým základným zoznamom predvolene povolených aplikácií, aby systém dokázal korektne fungovať i v takomto prípade, avšak takéto rozhodnutie by bolo potrebné dôkladne zvážiť, a to predovšetkým spektrum predvolene povolených aplikácií, ktoré, ak by bolo príliš široké, by mohlo znamenať potenciálnu bezpečnostnú hrozbu alebo hrozbu problémov so stabilitou. Alternatívne by aplikácia mohla pravidelne sťahovať zo serveru zoznam všetkých povolených aplikácií pre daného používateľa; týmto by sa však spomalila reakcia systému na zmeny, ako by aj bolo potrebné riešiť bezpečnostnú politiku dĺžky platnosti takýchto zoznamov. Tiež by pribudla hrozba editácie zoznamov inou, potenciálne škodlivou aplikáciou, čím by sa podstatne znížil bezpečnostný prínos navrhovaného systému.

3.2.4 Driver

Komponentom slúžiacim na monitorovanie systému a detekciu spúšťania aplikácií je aplikácia AMBS_LL[6]. Aplikácia je naprogramovaná v C a Assembleri, kompilovaná prostredníctvom Windows Driver Development Kit (WDDK). Výhodou tohto kompilátora je možnosť jednoduchej kompilácie driveru pre viacero platforiem, a teda i táto voľba umožňuje jednoduchšie rozšírenie navrhovaného systému na ďalšie operačné systémy Windows. Driver AMBS_LL volania systémovej API funkcie NtCreateSection() zasiela na ďalšie spracovanie aplikácií AMBSLL_DCom prostredníctvom rozhrania popísaného v 3.2.3. Prostredníctvom spomínaného rozhrania prijíma i rozhodnutie navrhovaného systému o povolení či nepovolení spustenia danej aplikácie; v prípade povolenia driver ukončí blokovanie spúšťania aplikácie, v opačnom prípade vráti negatívnu návratovú hodnotu STA-

[6]LL ako skratka pre Low Level

TUS_ACCESS_DENIED. Tento spôsob ukončenia aplikácie vyvoláva aj chybové hlásenie operačného systému Windows o pokuse o spustenie neplatnej aplikácie, ktoré by v prípade potreby mohlo byť v konkrétnych implementáciach odstránené zrejme napríklad hookovaním mechanizmu hlásenia chýb operačného systému. Driver bol vytvorený vhodnou modifikáciou návrhu uvedeného v [2], ale aj [10].

3.3 Ďalší vývoj a zhodnotenie rizík

V časti 3.2 sme uviedli návrh aplikácie, ktorá by mala po úpravách pre konkrétne podmienky nasadenia byť užitočným pomocníkom pre administrátorské tímy stredných a väčších spoločností rôznej štruktúry i granularity. Keďže ide o návrh novej aplikácie pre ktorú reálna plnohodnotná náhrada v praxi nebola nájdená a k jej reálnemu nasadeniu v praxi ešte nedošlo, rozhodne možno nájsť veľa parametrov a čŕt, ktoré by sa dali doplniť, upraviť či vymeniť. Tiež je pred ostrým využitím projektu nevyhnutné zhodnotiť riziká prevádzky systému a uviesť opatrenia vedúce k ich minimalizácií.

3.3.1 Potenciálne riziká

Vzhľadom na to, že aplikácia obsahuje komponent bežiaci v režime jadra, je dôležité si uvedomiť, že tento komponent predstavuje veľkú silu i veľkú slabinu aplikácie. Umožňuje totiž efektívne a takmer úplne zamedziť nedetegovaným pokusom o spustenie aplikácie, ale, na strane druhej, v prípade zlyhania môže systém úplne znefunkčniť, v horšom prípade viesť k chybe a spôsobiť stratu neuložených dát a informácií. Rizikovým je aj z pohľadu kolízií; i keď sa počas testovania žiadne kolízie neobjavili, z povahy driveru vyplýva, že by potenciálne mohol kolidovať s inými zabezpečovacími systémami používanými na danej počítačovej stanici.

Potenciálne nebezpečným je i previazanosť aplikácie na externú databázu; jednak môže dôjsť k jej zneprístupneniu, jednak prostredníctvom metód známych ako DNS poisoning, príp. inými úmyselnými alebo neúmyselnými obmedzeniami pripojenia. I keď v dnešnej praxi je počítač zvyčajne pripojený k sieti bez prestávky, je dôležité sa týmto rizikom zaoberať. Návrhy na jeho nápravu uvedieme v sekcii 3.3.2.

Ďalším potenciálnym rizikom pri neúplnom odladení navrhovaného systému pre danú platformu by mohla byť nedostatočná alebo problematická synchronizácia komponentov, nedostupnosť servera AMBS_AA či neočakávané chyby a zlyhania niektorého z komponentov navrhovaného systému.

Samostatnou kapitolou rizík by mohli byť úmyselné pokusy o napadnutie aplikácie softvérom tretích strán. K výhode navrhovaného systému v takejto situácii patrí prítomnosť kódu na úrovni režimu jadra, kde môžu byť doplnené efektívne seba ochranné mechanizmy pre všetky komponenty systému. Zaujímavým zvýšením bezpečnosti by bolo i zabezpečenie interných komunikačných kanálov aplikácie, ktoré sú graficky naznačené na obrázku 3.4. Konkrétne, bolo by možné rozšíriť použitie socketov o SSL alebo iný spôsob zabezpečenia, ktorý by zabezpečil (nad rámec zabezpečení protokolu TCP) nielen doručenie dát, ale aj ich autentickosť či overenie pôvodu. Tiež by bolo rozumné overovať identifikáciu serveru rozhodujúceho o prítomnosti alebo neprítomnosti vzorky danej aplikácie v databáze; toto rozhodnutie je totiž pre beh navrhovaného systému kritickým. Avšak, v náplni navrhovaného systému nie je chrániť pred malwarom - preto je dôležité, aby bol na stanici prevádzkovaný spoľahlivý zabezpečovací systém proti malware. Navyše, vzhľadom na špecifickosť a originalitu systému nemožno predpokladať, že by k podobným cieleným útokom došlo, kým navrhovaný systém nie je všeobecne rozšírený.

Ako je už v svete zabezpečovacích aplikácií overeným faktom, častým rizikom sú aj nekvalifikované zásahy používateľa do behu systému. Týmto rizikám možno predísť predovšetkým dobrým nastavením používateľských práv na koncovej stanici, ale aj školeniami o používaní produktu.

Keďže navrhovaný systém musí v momentálnej verzii pri každom pokuse o spustenie aplikácie komunikovať s externou databázou, obzvlášť v prípade príliš vyťaženej siete či pripojenia cez VPN na úzkej dátovej linke môže dôjsť k signifikantnému spomaleniu behu počítača pri intenzívnej manipulácií s rôznymi aplikáciami. Pri dnešných broadband pripojeniach a bežnom spôsobe používania osobných počítačov, kde sa aplikácia spustí a dlhšiu dobu používa, by to však nemal byť dôležitý problém. Pri testovaní na nezaťaženej lokálnej sieti s teoretickou rýchlosťou 100 Mbps spomalenie nebolo zaujímavé a pri bežnom používaní systému ho používatelia nespozorovali vôbec.

Potenciálnym rizikom je aj útočníkom vykonané úmyselné nahradenie niektorého z komponentov iným. V prípade nahradenia driveru by došlo prakticky k znefunkčneniu systému, ktoré by bolo detegované iba v prípade nekorektnej komunikácie s komunikačným komponentom. Nahradenie komunikačnej aplikácie by mohlo viesť napríklad k povoleniu všetkých aplikácií alebo zakázaniu žiadaných aplikácií, čo by opäť bolo detegované iba nekorektnou komunikáciou s niektorým z ďalších dvoch komponentov. Ak by však tento komponent nebol nahradený, ale iba pridaný a pokúsil sa o komunikáciu s GUI komponentom, tak by k ohrozeniu zrejme nedošlo; komponent AMBS_AA totiž implementuje server,

ktorý môže korektne odpovedať na požiadavky viacerých klientov. Nahradenie GUI by mohlo znamenať podobné následky ako nahradenie komunikačnej aplikácie. Ako odpoveď na uvedené hrozby by sme mohli ponúknuť napríklad kontrolovanie kontrolných súčtov, respektíve hašov aplikácií pri otváraní komunikačného kanálu; ide o riešenie pomerne jednoducho implementovateľné a pomerne efektívne riešiace uvedenú potenciálnu hrozbu. Takéto haše by zrejme boli nastavené priamo pri kompilovaní komponentov a výmena alebo update jedného z komponentov by znamenala nutnosť zmenu ďalšieho komponentu, pretože ich uloženie v externom súbore by znamenalo výskyt hrozby jednoduchého prepísania takejto hodnoty; takéto prepísanie je potenciálne možné aj v kóde aplikácie, a to napríklad driverom bežiacim v režime jadra.

Stav rizík sa pokúsme zhrnúť nasledovnou tabuľkou. Označme P pravdepodobnosť, že k ohrozeniu príde, a rozdeľme ju na 3 úrovne (nízka = 1, stredná = 2, vysoká = 3). Úroveň nebezpečenstva rizika označme N a zaveďme 3 úrovne nebezpečenstva (nízke = 1, stredné = 2, vysoké = 3). Následne pridajme odhad možnosti efektívnej minimalizácie rizika - tieto možnosti minimalizácie budú ďalej opísané v sekcii 3.3.2. Sumárne hodnotenie neuvádzame; bolo by síce možné použiť niektorý zo štandardných prepočtov na celkové hodnotenie hrozby, avšak v prípade navrhovaného systému nejde o hodnotenie systému určeného na nasadenie, ale skôr o diskusiu o možnostiach a potrebe minimalizácie rizík. Ich pravdepodobnosť a nebezpečenstvo tiež závisí na konkrétnych podmienkach nasadenia u konkrétneho klienta.

Riziko	**P**	**N**	**Možnosť minimalizácie**
Fyzický útok	1	3	čiastočne
Nahradenie komponentu falošným	1	3	áno
Nedostupnosť databázy	2	2	čiastočne
Softvérová kolízia	2	3	iba testovaním
Spomalenie systému	1	1	čiastočne
Útok malware	1	3	čiastočne
Zásah používateľa	1	2	áno, školenia
Zlyhanie komponentu	1	3	čiastočne

3.3.2 Ďalší vývoj a vylepšovanie

Na základe rizík analyzovaných v sekcii 3.3.1, ale aj na základe testovania navrhovaného systému a konzultácií s IT odborníkmi z praxe sa objavili možnosti ďalších úprav a rozšírení aplikácie. Tieto rozšírenia v tu prezentovanom návrhu nie sú implementované z dôvodu

častej previazanosti na konkrétne podmienky spoločnosti, kde by sa systém využíval.

Jednou z už uvedených optimalizácií je výmena implementácie databázy za inú databázu, ktorá by plne podporovala všetky podmienky na databázu kladené v 3.2.1. Vplyvom tohto kroku by sa nielen podarilo databázu úplne normalizovať, ale aj znížiť režijné náklady na administráciu skupín a možno použiť štandardné, implementátorom databázy navrhnuté riešenie. Prechod na takúto databázu by ovplyvnil iba komponent AMBS_AA, kde by išlo o zmenu správy tabuliek (po zmene už iba jedinej tabuľky) uchovávajúcich povolené aplikácie.

Rozšírením v prípade záujmu implementovateľným pomerne jednoducho je uchovávanie aktuálne povolených aplikácií lokálne pre umožnenie práce offline, teda bez možnosti pripojiť sa k databázovému serveru. V prípade takéhoto riešenie by bolo nutné vhodne zvoliť režim a spôsob aktualizácie dát z databázy a ich platnosť. Tiež by však pribudol i problém, ktorý sme pri našom riešení mohli zanedbať - a to pokus napadnúť tieto definície potenciálnym útočníkom a pridať sem tretiu aplikáciu. Treba však pripomenúť, že tento útočník by musel útok viesť nepriamo, teda napr. svoj skript spustiť prostredníctvom inej, povolenej aplikácie.

Pre skrátenie času odozvy administračného tímu na potrebu zmien či riešenie chýb by mohlo byť pre istú časť potenciálnych klientov užitočné aj priame previazanie na informačný systém IT oddelenia; takúto väzbu však nebolo možné navrhnúť v čase návrhu systému, keďže takéto systémy sú špecifické pre každého klienta. Implementácia uvedeného návrhu by ovplyvnila pravdepodobne iba komponent AMBS_AA. Užitočnou by tiež mohla byť implementácia akejsi obálky okolo navrhovaného systému, ktorá by štandardizovala komunikáciu s uvedenými špecifickými administračnými a kontrolnými systémami.

Ďalšou zaujímavou možnosťou by bolo pokračovať vo vývoji driveru a pridať mu ďalšiu funkcionalitu. Užitočným by mohlo byť napríklad odchytávanie pokusov o inštaláciu aplikácií miesto pokusov o spustenie. Toto však prináša už na prvý pohľad mnoho problémov s analýzou možných spôsobov inštalácie a ich korektným rozpoznávaním; niektoré programy navyše na inštaláciu nevyužívajú štandardné prostredia a frameworky, ale používajú osobitne vyvinuté inštalátory. Preto by zrejme bolo nevyhnutné implementovať heuristické metódy na detegciu takýchto pokusov. Keďže driver deteguje volania funkcie NtCreateSection, nie je celkom pravdou, že deteguje spúšťané procesy. Táto API funkcia sa totiž používa na vytváranie blokov virtuálnej pamäte pre dané vlákno; je teda bežné, že je použítá aj z iného dôvodu, ako je vytvorenie procesu. V princípe nám tento rozdiel neprekáža, pretože tieto volania sú tiež filtrovateľné pomocou implementovaných whitelis-

tov; avšak, ak by to bolo dôležité, bolo by možné hookovať volania všetkých API funkcií potrebných pre spustenie aplikácie a na základe komplexnej správy z ich volaní rozhodnúť, či ide o spúšťanie procesu. Takto implementovaný systém by bol taktiež možno imúnnejší voči útokom typu KHOBE, ktoré spomenieme neskôr.

Keďže momentálna verzia sleduje iba spúšťané aplikácie, neexistuje filtrovanie na základe parametrov spúšťaných súborov; práve sledovanie takýchto parametrov by mohlo byť zaujímavé z bezpečnostných dôvodov, a to napríklad pri spúšťaní prezentácií, otváraní archívov, atp. Takáto zmena by pravdepodobne ovplyvnila minimálne komponenty AMBS_LL a AMBS_AA. Uvedený prístup by si však vyžadoval skôr prístup formou blacklistov a do veľkej miery sa problémy zabezpečenia v tomto smere pokúšajú riešiť antivírusové systémy.

Vzhľadom na najnovšie trendy a rôzne metódy potenciálnych útokov, ktorých príkladom by mohol byť článok [11] o útoku KHOBE[7], je zaujímavým nápadom na zamyslenie i využitie iných metód ako je hookovanie API funkcií systému na riadenie spúšťania aplikácií. Nové operačné systémy Windows (Windows Vista, Windows 7) poskytujú nové metódy zasahovania do systémových funkcií, ktoré vznikli ako priama podpora pre antivírusové systémy a sú využívané napríklad produktom Microsoft Security Essentials. Na druhej strane si však treba uvedomiť, že takýto útok by bol pravdepodobne úspešný v prípade, že by útočný softvér bol v systéme umiestnený pred nami navrhovaným systémom, alebo spustený prostredníctvom automaticky spúšťaného driveru systému Windows pred systémom AMBS. Vzhľadom na nevysokú pravdepodobnosť úspešnosti takéhoto útoku, ako aj zvolený operačný systém Windows XP, sa však v čase návrhu neponúkali žiadne riešenia, ktoré by sme pred použitou metódou hookovania API funkcií preferovali. Avšak v prípade rozšírenia systému na ďalšie operačné systémy Windows by mohlo byť zaujímavé sa s týmito novými metódami, ako napríklad File System Minifilter Drivers, oboznámiť a prípadne implementovať komponent AMBS_AA pomocou takýchto funkcií.

Nepochybne vyzývajúcou myšlienkou je rozšíriť kompatibilitu navrhovaného systému na ďalšie operačné systémy. Rozšírenie na systém Windows 2000 či iné 32-bitové verzie Windows by nemalo byť v princípe problematické, i keď treba zvážiť dopady novo implementovaných systémov rozšíreného zabezpečenia vo verziách Windows Vista a Windows 7. Ako problematické sa ukazuje rozšírenie na 64-bitové platformy Windows[8]. Nie len že dochádza k zmene veľkosti smerníkov, základných dátových typov a množstva adresovate-

[7] I keď na tento problém bolo upozornenie už dávnejšie, a to napríklad v [1], [8]

[8] Podľa [6], kapitola 5

ľnej pamäte až na 16 EB[9], ale je nutné využívať aj iné API funkcie, ktorých presný popis je síce formálne k dispozícii, ale ich využívanie programátormi sa ešte iba začína rozvíjať.

[9]EB = exabyte; 16 EB = 2^{64}B, podľa [6], kapitola 5, strana 132

Záver

V práci sme sa pokúsili navrhnúť systém na efektívnu detekciu, analýzu a riadenie spúšťania aplikácií v prostredí operačného systému Microsoft Windows SP3. Vzhľadom na uvedené parametre, a to menovite modularitu, centralizáciu, jednoduchosť administrácie, prehľadnosť, spoľahlivosť a rozšíriteľnosť, sa nám podarilo vytvoriť jedinečný produkt, ktorý uvedenými vlastnosťami predčí dnes dostupné riešenia pokúšajúce sa o riešenie čiastkových či podobných problémov v možnosti nasadenia vo vybranej cieľovej skupine, ktorou sú potenciálne stredné a väčšie spoločnosti, prípadne školy či iné zariadenia vyžadujúce efektívne riadenie povoľovania spúšťania aplikácií. Ako jeden zo signifikantných prínosov práce možno uviesť jej veľmi vysokú modularitu, ktorá umožňuje rýchle a bezproblémové zmeny funkčnosti, ale aj celých komponentov. Tiež použitím značne multiplatformálneho prístupu pri vývoji komponentov, ktoré nie sú priamo previazané na fungovanie daného operačného systému, sme umožnili jednoduchšiu a bezproblémovejšiu prenositeľnosť na počítače využívajúce modernejšie alebo z iného dôvodu alternatívne operačné systémy. Riešenie lokalizácie aplikácie umožňuje spraviť navrhovaný systém užívateľsky prijateľným pre každý národ, etnikum či krajinu, a to bez nutnosti upravovať centralizované časti systému. Už uvedená centralizácia tiež umožňuje efektívnu administráciu systému a možnosť jeho previazania s inými existujúcimi systémami v danej spoločnosti. Jednoduchá a transparentná manažovateľnosť trivializuje vyvodzovanie zodpovednosti za problémy, keďže je zrejmé, ktorí používatelia môžu vykonávať dané zmeny. Nepochybne dôležitou je i otázka bezpečnosti; tej sme sa okrem iného venovali v časti 3.3.1; použitím štandardných riešení alebo vďaka modularite jednoduchému nasadeniu alternatívnych riešení sa systém stáva nie len spoľahlivým, ale aj dynamickým a ľahko reagujúcim na výzvy či zo strany používateľov alebo vonkajších hrozieb. Aby sme naznačili previazanosť produktu na prax, citujme vyjadrenia niekoľkých odborníkov v sektore IT podpory uvažujúcich o ďalšom potenciálnom využití tejto práce:

Miroslav Hrubjak, Manager IT, GroupM: *„AMBS je systém podstatne znižujúci zaťaženie IT odelenia v ľubovoľnej väčšej spoločnosti. Jeho prispôsobenie konkrétnym potrebám*

toho ktorého podniku a následné nasadenie by mohlo priniesť zaujímavé benefity.“

Petr Turek, Referent informačních a komunikačních technologií (servers, security) - Státní správa ČR: *„AMBS je jeden z mnoha systémů, který reflektuje na potřeby integrace užití TPE ve strukturách menších i větších firem/podniků. Podpora takovýchto aplikací na trhu v ČR i SR je velmi malá a skromné množství dodavatelů, nabízí pouze neucelené a hlavně musím zdůraznit, velmi nestabilní kusy SW, které se používají, ale většina koncových implementačních skupin tato řešení z důvodu nestability používat přestává. Jsou tu jisté možnosti užití software designovaného na míru, ale to za vysokých pořizovacích či provozních nákladů. AMBS přichází včas s vhodným chybějícím řešením, a lze si v budoucnu bez obav představit možnou integraci do určitých sektorů státní správy. Jako výhodu bych si dovolil uvést velmi dynamický přístup aplikace a autora směrem k užívání a modifikacím SW i jeho konfigurace, předpokládané nízké pořizovací náklady a plnou lokalizaci na domácí SR i ČR trh.“*

Ako sme uviedli už pri formulácii cieľov práce, cieľom nebolo vytvoriť produkt priamo nasaditeľný v praxi, ale istý návrh, šablónu, ktorá po určitých personalizáciach, lokalizáciach, rozšíreniach a úpravách bude v praxi použiteľná. Vďaka vysokej modularite, ale aj štandardným postupom a štruktúram, ktoré sme sa snažili v programovom kóde aplikácie aplikovať, systém uvedené požiadavky spĺňa dostatočne, ako potvrdzujú aj vyjadrenia odborníkov z praxe. Nepochybne, ako v každej inej aplikácií podobne veľkého systému, i tu sa vyskytli mnohé problémy, riziká a kompromisy. Niektoré otázky sa vynorili už pri počiatočnej špecifikácii návrhu systému; pokúsili sme sa ich zhrnúť a použité riešenia odôvodniť v časti 1. Mnohé z problémov a kompromisov sme spolu s riešeniami načrtli v časti 3.2, analyzačný pohľad na riziká sme aplikovali v 3.3. Možno preto uviesť, že naša snaha poskytnúť komplexný návrh systému umožňujúci rýchle zorientovanie sa v problematike potenciálnym ďalším implementátorom systému, bola naplnená.

Literatúra

[1] Andrey Kolishak: TOCTOU with NT System Service Hooking, 30.12.2003, dostupné na internete: http://seclists.org/bugtraq/2003/Dec/351 dňa 19.5.2010

[2] Anton Bassov: Hooking the native API and controlling process creation on a system-wide basis, 18.10.2005, dostupné na internete: http://www.codeproject.com/KB/system/soviet_protector.aspx dňa 19.5.2010

[3] David B. Probert, Ph.D.:Windows Kernel Internals, Process Architecture, 10.6.2009, dostupné na internete: http://www.scribd.com/doc/20689966/Windows-Kernel-Internals-Process-Architecture dňa 19.5.2010

[4] Ivo Ivanov: API hooking revealed, 3.12.2002, dostupné na internete: http://www.codeproject.com/kb/system/hooksys.aspx dňa 19.5.2010

[5] Jeffrey Richter, Christophe Nasarre: Windows Via C/C++ (Fifth edition), Microsoft Press 2008. ISBN 978-0-735-62424-5

[6] Johnson M. Hart: Windows System Programming (Fourth Edition); Pearson Education, Inc. 2010. ISBN 978-0-321-65774-9

[7] Keith Vertanen, Cavendish Laboratory, University of Cambridge: Java / C++ socket class, Október 2009, dostupné na internete: http://www.keithv.com/software/socket/ dňa 19.5.2010

[8] Matt Bishop, Michael Dilger: Checking for Race Conditions in File Accesses; University of California at Davis 1995

[9] Michael Kifer, Arthur Bernstein, Philip M. Lewis: Database Systems (Second Edition),An Application-Oriented Approach, Addison Wesley 2005. ISBN 978-0-321-22838-3

[10] Ms-Rem: Perehvat API funkcij v Windows NT (časť 3). Nulevoe koľco., 16.5.2005, dostupné na internete: http://www.wasm.ru/article.php?article=apihook_3 dňa 19.5.2010

[11] Skupina Matousec: KHOBE – 8.0 earthquake for Windows desktop security software, 5.5.2010, dostupné na internete: http://www.matousec.com/info/articles/khobe-8.0-earthquake-for-windows-desktop-security-software.php dňa 19.5.2010

[12] Stanley Wang: Using Memory Mapped Files and JNI to communicate between Java and C++ programs, 19.6.2002, dostupné na internete: http://www.codeproject.com/KB/java/sharedmem_jni.aspx dňa 19.5.2010

[13] Toby Opferman: Driver Development, Part 1: Introduction to Drivers, 6.2.2005, dostupné na internete: http://www.codeproject.com/KB/system/driverdev.aspx dňa 19.5.2010