VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

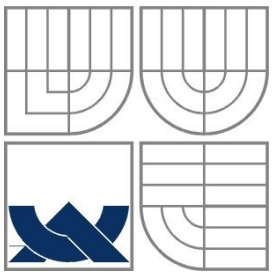
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

ANALÝZA REGISTRŮ MICROSOFT WINDOWS

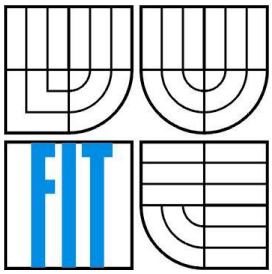
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE Bc. MIROSLAV HULA
AUTHOR

BRNO 2011



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

ANALÝZA REGISTRŮ MICROSOFT WINDOWS

MICROSOFT WINDOWS REGISTRY ANALYSIS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. MIROSLAV HULA

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. PETR CHMELAŘ

BRNO 2011

Abstrakt

Poznání a práce s registry operačního systému Microsoft Windows je z pohledu bezpečnosti důležitou schopností. Tuhle schopnost využívá jak škodlivý software, tak i software pro opravu poškozených registrů, u kterých poškození vzniklo právě činností škodlivého softwaru. Aplikace pro přístup a práci s registry jsou však závislé na platformě, co nemusí být vždy výhodné a může to vézt k dalším problémům, když tato platforma není bezpečná. Proto je cílem této práce vytvořit aplikaci pro přístup a práci s registry, která bude nezávislá na platformě a umožní analyzovat vlivy škodlivého softwaru na ně.

Abstract

Understanding and working with Microsoft Windows registry is an important ability from the perspective of security. This ability is used by malicious software as well as by software, which repaires the damage caused by activity of malicious software. However, applications accessing and working with the registry are platform dependent, which may not always be convenient and it can lead to other problems if the platform is not secure. Therefore, the aim of this work is to create a platform independent application for accessing and working with registry, which makes possible to analyse the effect of malware on registry.

Klíčová slova

Registry operačního systému Microsoft Windows, škodlivý software, INI soubory, Regedit, Wine, klíč, hodnota, bunka nk, bunka vk, bunka sk, bunky lh, lf, li, ri, úprava registrů malwarem

Keywords

Microsoft Windows Registry, malware, INI files, Regedit, Wine, key, value, nk cell, vk cell, lh, lf, li, ri cells, malware registry modification

Citace

Hula Miroslav: Analýza registrů Microsoft Windows, diplomová práce, Brno, FIT VUT v Brně, 2011

Analýza registrov Microsoft Windows

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Petra Chmelaře. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Miroslav Hula
16.mája 2011

Poďakovanie

Rád by som poďakoval vedúcemu práce Ing. Petrovi Chmelařovi za jeho pripomienky a odborné rady.

Obsah

1 Úvod

Neustála informatizácia spoločnosti a rozširovanie internetu sú neprehliadnuteľnými procesmi, ktoré možno v dnešnom svete pozorovať. Zefektívnenie práce, zlepšenie dostupnosti informácií, či prepojenie ľudí a zdrojov z rôznych kútov sveta sú len časťou z nespočetného množstva výhod, ktoré tieto procesy so sebou prinášajú. Každá minca má však dve strany a proces rozvoja informatiky nie je žiadnou výnimkou.

Informatizácia spoločnosti so sebou prináša čoraz väčšie množstvo informácií v digitálnej podobe, pričom sa často jedná o dôverné informácie. V spojení s rýchlym rozširovaním internetu a jeho dostupnosťou v čoraz väčšej časti sveta, vzniká opodstatnená hrozba straty alebo zneužitia dát. Strata alebo zneužitie dát užívateľov nie sú žiadnym novým fenoménom, je však pravdepodobné, s prihliadnutím na trendy v spoločnosti, že časom bude väčším problémom dáta skryť ako ich nájsť. To by mal byť dôvod, prečo by mala byť bezpečnosti a prevencii spomenutých hrozieb venovaná zvýšená pozornosť.

Rizík, ktoré ohrozujú dáta, počnúc hackermi a končiac škodlivým softwarom, je nespočetne mnoho a rozsah jednej práce nedokáže popísať celú túto problematiku. Preto je táto diplomová práca zameraná na špecifickú oblasť súvisiacu s bezpečnosťou, na oblasť často využívanú škodlivým softwarom, a to konkrétne na registre operačného systému Microsoft Windows.

Registre operačného systému Microsoft Windows sú úložiskom množstva dát a dôležitých nastavení. A to je dôvod, prečo sú zaujímavé aj z pohľadu škodlivého softwaru. Neodborná manipulácia s nimi, či ich prílišné poškodenie však vedú k nefunkčnosti celého operačného systému, ktorý je na nich existenčne závislý. Riešením tohto nežiaduceho stavu je preinštalovanie celého systému alebo analýza a oprava poškodených registrov z iného počítača. Ako analýza, tak aj oprava sú v dnešnej dobe vykonateľné pomerne jednoducho, avšak opäť iba z prostredia operačného systému Microsoft Windows. Tento fakt vedie k mnohým problémom a vynúteným zmenám operačných systémov najmä z pohľadu administrátorov počítačových systémov či správcov sietí. Hlavný problém možno rozdeliť na dve časti. Prvou je fakt, že medzi bežnými užívateľmi pretrváva a prevláda obľúbenosť operačného systému Microsoft Windows, ktorý je pre nich jednoduchší a sú s ním zvyknutí pracovať. Na druhej strane, medzi administrátormi počítačových systémov a správcami sietí je vyšší záujem o serverové operačné systémy založené na Unixe, keďže ich stabilita, bezpečnosť a čas bezporuchovosti je vyšší než pri serverových operačných systémoch Microsoftu. Tu teda vzniká rozpor, či používať na počítačových staniciach aj na serveroch operačné systémy Windows na úkor stability a dostupnosti služieb ponúkaných serverom a mať v podstate bezproblémový spôsob opravy systému na prípadne poškodenej pracovnej stanici. Alebo používať na pracovných staniciach operačný systém Windows a na servery unixový operačný systém, čo bude viesť k vyššej stabilite, bezpečnosti a dostupnosti služieb, ale možnosť analýzy poškodenia staníc, prípadne ich opravenia

bude značne obmedzená. Z dôvodu nízkej znalosti používania operačných systémov na báze Unixu medzi užívateľmi nie je potrebné takéto alternatívy riešenia uvažovať.

Z vyššie uvedeného je zrejmé, že problematika bezpečnosti je naozaj obsiahla a pri hľadaní optimálnych riešení je často potrebné hľadať kompromisy, ktoré sú však často viac orientované na spokojnosť užívateľa, než na bezpečnosť. Preto cieľom tejto práce je vytvoriť aplikáciu, ktorá bude schopná analyzovať registre operačného systému Microsoft Windows nezávisle na tomto operačnom systéme a teda aj z operačných systémov založených na Unixe. Okrem toho by mala umožňovať porovnávanie registrov pred a po napadnutí škodlivým softwarom. Výsledky týchto porovnaní budú spracované a analyzované s cieľom odhaliť vplyvy škodlivého softwaru na registre operačného systému.

Druhá kapitola tejto práce vysvetľuje pojem „škodlivý software“ a venuje sa popisu jeho druhov a tiež ich správaním sa v systéme, prípadne ich vplyvom na registre operačného systému Microsoft Windows.

Ako už bolo spomenuté, registre slúžia ako úložisko množstva konfiguračných dát a nastavení v operačnom systéme Windows. Presnejším popisom uloženia týchto dát, ich prepojeniami, súvislosťami, významom a vzťahmi, rovnako ako aj krátkym zhrnutím ďalších spôsobov uložení dát, sa zaoberá tretia kapitola. V tejto kapitole je zároveň venovaná pozornosť už existujúcim prístupom k registrom systému Windows. Ďalej sú tu popísané základné princípy prístupov a ich výhody, či nevýhody. Väčšia pozornosť je venovaná nástroju Regedit, ktorý slúži k prístupu a práci s registrami v operačnom systéme Windows a API funkciám, ktoré na tento účel využíva.

Nakoľko pre účely aplikácie bude potrebné získaný obsah registrov uložiť a neskôr s ním pracovať, prípadne spracovávať výstup analýzy registrov, je potrebné si správne určiť dátový model pre uloženie týchto registrov. Tomu však predchádza analýza registrov a detailný popis uloženie jednotlivých prvkov registrov, ich prepojení a vzťahov medzi nimi, bez ktorého by nebolo možné navrhnúť správny dátový model. Návrhom a popisom tohto modelu ako aj samotnou analýzou spôsobu uloženia registrov sa zaoberá štvrtá kapitola.

Hlavné body implementácie a najdôležitejšie časti implementovanej aplikácie sú popísané v piatej kapitola. Taktiež sú tu popísané najzávažnejšie problémy, ktoré sa pri implementácii objavili ako aj ich pôvod a dôvody existencie.

Ďalšia, šiesta kapitola, sa venuje experimentom, ktoré boli vykonané s použitím zhotovenej aplikácie a ktorých cieľom je odhaliť vplyv konkrétnych škodlivých softwarov na registre.

Siedma kapitola obsahuje kvalitatívne porovnanie navrhnutého a implementovaného riešenia s existujúcimi riešeniami. Taktiež sú v tejto kapitole popísané návrhy rozšírení, ktoré by mohli byť z hľadiska tejto diplomovej práce prospešné či už pre rozšírenie funkčnosti implementovanej aplikácie alebo celkovo pre oblasť bezpečnosti.

Záver zhodnocuje dosiahnuté výsledky a získané poznatky.

2 Malware

Pod pojmom malware rozumieme škodlivý software, presnejšie software, ktorý je určený k poškodeniu počítačového systému alebo k vniknutiu do neho a vykonávaniu neautorizovaných činností bez vedomia užívateľa. Takýchto typov softwarov je viac druhov a pojem malware slúži len ako ich všeobecné označenie. Konkrétne sa jedná o počítačové vírusy, trójske kone, rootkity, červy, spyware, adware a iné. Nasledujúce podkapitoly sa venujú najdôležitejším z nich a následne je popísaný ich vplyv na registre operačného systému Microsoft Windows.

2.1 Vírus

Vírus je škodlivý software, ktorý je spravidla súčasťou iného súboru, so schopnosťou sebareplikácie a s cieľom infikovať časti operačného systému alebo aplikačných programov [1].

Preto, aby vírus mohol vykonávať svoju činnosť musí byť spustený. Keďže žiadny z užívateľov by ho zrejme dobrovoľne nespúšťal, musí byť vírus súčasťou iného súboru. A to buď spustiteľného, vykonateľného pomocou nejakej aplikácie, ako napríklad Microsoft Word alebo musí byť súčasťou systémovej oblasti disku. V okamihu, keď je tento súbor spustený alebo vykonaný, vykonáva sa bez vedomia užívateľa kód vírusu a vírus sa snaží zabezpečiť sebareplikáciu pripojením resp. infikovaním ďalších vhodných súborov, ktoré poslúžia k jeho spusteniu. Až po činnostiach, ktoré vykonával vírus, medzi ktoré okrem sebareplikácie môže patriť napríklad aj poškodzovanie a ničenie dát v počítači, sa spúšťa vykonanie samotného súboru.

Zaujímavými a veľmi dôležitými vlastnosťami vírusov sú šifrovanie, oligomorfizmus, polymorfizmus a metamorfizmus vírusov. Účelom šifrovania je spraviť kód vírusu neprehľadným a tým sťažiť jeho analýzu a prípadné odhalenie antvírusovými aplikáciami. Tento spôsob šifrovania realizovali niektoré druhy vírusov zašifrovaním samotného kódu vírusu nejakým jednoduchým algoritmom s tým, že pred vírusom sa vždy nachádzala dešifrovacia časť, ktorá zabezpečovala transformáciu kódu do pôvodnej podoby. Vírusy šifrované týmto spôsobom majú takmer konštantnú podobu a ich detekcia je možná dokonca aj bez potreby dešifrovať zašifrovaný kód vírusu. K tejto detekcii plne postačovala dešifrovacia časť vírusu.

Postupom času sa prešlo k spôsobu, pri ktorom sa menila dešifrovacia časť vírusu. Tento typ vírusov, nazývaných oligomorfné, už teda nebolo možné detekovať na základe dešifrovacej časti. Pri ich odhaľovaní bolo potrebné sledovať aj kód samotného vírusu, ktorý bolo nutné predtým dešifrovať.

Polymorfný vírus je typ vírusu, ktorý dokáže vytvárať nekonečné množstvo dešifrovacích častí a tie používajú rôzne metódy šifrovania zdrojového kódu vírusu. Pri vírusoch využívajúcich túto vlastnosť sa líši každý exemplár a teda je veľmi problematické ho odhaliť, keďže medzi jednotlivými kópiami neexistujú žiadne zhodné sekvencie kódu. Je však potrebné si uvedomiť, že neexistujú

podobné sekvencie v kódoch preto, lebo ide o kódy zašifrované určitým algoritmom, ktorý je neznámy. Nejde však o zmenu samostatných kódov vírusov iba o zmenu šifrovania kódov vírusov.

Zásadne odlišnými od polymorfných vírusov sú vírusy metamorfné. Tie využívajú spôsob znemožnenia odhalenia vírusov, ktorý je založený na zmene zdrojového kódu samotného vírusu. Túto zmenu vykonáva samotný vírus a to tým, že prekladá svoj vlastný pseudokód, ktorý je jeho súčasťou, kompilátorom, ktorý buď nesie so sebou alebo si ho dohľadá, napríklad na disku užívateľa. Týmto prekladom vznikajú celkom odlišné kópie vírusu. Takýto typ vírusu nemá dešifrovaciu časť a žiadna časť tela kódu vírusu nie je konštantná [2].

2.2 Trójsky kôň

Trójsky kôň je druh škodlivého softwaru, ktorý na rozdiel od vírusov nie je schopný replikovať sám seba a spravidla vykonáva nejakú skrytú škodlivú činnosť. Nakoľko trojský kôň nie je schopný replikovať sám seba, k jeho rozšíreniu je potrebná určitá intervencia zo strany užívateľa. Môže ísť o kliknutie na nejaký internetový odkaz alebo o manuálne nainštalovanie trójskeho koňa pod dojmom, že ide o iný software, hlavne ak má súbor dvojitú príponu, ktorú užívateľ nespozoruje. Ďalšou cestou, ako sa tento druh škodlivého softwaru môže dostať do počítača, je prostredníctvom iného programu, napríklad vírusu, ktorý je schopný ho spustiť alebo nainštalovať. Keď už je trójsky kôň v systéme, spravidla vyčkáva na určitý impulz pre spustenie. Týmto impulzom môže byť spustenie užívateľom alebo ďalším programom, prípadne spustenie na základe jeho vlastného časovača, ktorý je závislý na špecifickom dátume a čase alebo na špecifickej udalosti ako napríklad počet spustení určitej aplikácie.

Samotný trójsky kôň sa často skladá z dvoch častí, a to z časti serverovej a časti klientskej. Serverová časť je časťou, ktorou sa infikuje počítač, kým klientská zostáva u útočníka. Keď je serverová časť spustená, sú pre útočníka otvorené tzv. zadné dvierka a on ich môže prostredníctvom klientskej časti jednoducho využiť, napojiť sa na serverovú časť trójskeho koňa a získať tým prístup k počítaču užívateľa a následne vykonávať v podstate ľubovoľnú činnosť. Okrem týchto funkcií umožňujú trójske kone vo všeobecnosti množstvo ďalších. Napríklad môžu vymazávať súbory na napadnutom počítači, sledovať stlačenia kláves a z nich získané dôverné informácie zasielať tretím stranám resp. útočníkom, udržujú zadné dvierka pre útočníkov pre využitie pri ďalších útokoch a v neposlednom rade napomáhajú šíreniu ďalšieho škodlivého softwaru alebo šíreniu nevyžiadanej pošty – spamu [3].

2.3 Červ

Červ je typ softwaru, ktorý sa dokáže replikovať zo systému na systém bez intervencie užívateľa alebo iných osôb a bez použitia softwaru, ktorého by bol súčasťou, čím sa líši od vyššie popísaných vírusov. Pri rozširovaní využíva typicky počítačovú sieť a šíri sa vo forme sieťových paketov, kým vírus sa šíri v podobe infikovaných súborov. Ak takéto pakety dorazia do systému so špecifickou bezpečnostnou dierou, môže dôjsť k replikácii červa a produkcii ďalších infikovaných paketov. Šírenie červov teda spočíva v zneužívaní určitých bezpečnostných dier operačných systémov alebo aplikácií a ich rozšírení medzi užívateľmi.

Z vyššie uvedeného je zrejmé, že fungovanie založené na princípe zasielania paketov patrí k nižšej sieťovej vrstve, než je tomu pri vírusoch, čo spôsobuje problémy pri detekcii tohto typu škodlivého softwaru. Z pohľadu systémového pracujú červy na úrovni operačného systému, ktorá je pre užívateľov zakázaná. Často sa potom stáva, že ich existencia je zaznamenaná až keď ich nekontrolovaná replikácia spotrebováva aj systémové zdroje a tým spomaľuje alebo zabraňuje činnosti iných aplikácií prípadne, keď spotrebováva veľké množstvo sieťového prenosového pásma [3].

2.4 Rootkit

Rootkit je druh softwaru, ktorý sa snaží zamaskovať vlastnú prítomnosť v počítači a prípadné činnosti, ktoré v ňom vykonáva. Pôvodne ide o druh softwaru, ktorý sa objavoval iba na počítačoch s operačnými systémami založenými na Unixe. V nich však došlo k zmenám a opravám hlavne systémových knižníc a systémových programov, ktorých sa tento problém najviac týkal a v súčasnosti je problém rootkitov prevažne problémom operačných systémov Microsoft Windows [3].

Činnosti, ktoré sa rootkit snaží zamaskovať sú často spojené so sieťovou aktivitou, skrývaním súborov a procesov, alebo so zmenami registrov. Keďže práve problematika registrov je hlavnou témou tejto práce, budeme sa tejto časti venovať podrobnejšie.

Rootkity možno rozdeliť podľa vrstvy, na ktorej pracujú na *user mode* rootkity a na *kernel mode* rootkity. *User mode* rootkity zahŕňajú skrývanie na úrovni užívateľského alebo aplikačného pamäťového priestoru. V podstate to znamená, že hocikedy, keď aplikácia vyvolá nejaké systémové volanie, jeho vykonanie štandardne sleduje dopredu definovanú cestu, tak rootkit môže toto volanie využiť v hociktorej z častí na jeho ceste. Najbežnejším príkladom zneužitia na tomto móde je modifikácia systémových knižníc DLL v pamäti. Napríklad, aplikácia požaduje získanie zoznamu kľúčov v registroch systému. Preto, aby tento zoznam mohla dostať, zavolá aplikácia funkciu, ktorá jej zoznam vráti. Kód tejto funkcie je uložený v jednom z množstva súborov systémových knižníc DLL. Zatiaľ rootkit pracujúci na vrstve *user mode*, nájde umiestnenie volanej funkcie a modifikuje ju

tak, aby keď dôjde k vykonaniu funkcie, bolo jej vykonanie presmerované na kód samotného rootkitu. Rootkit potom sám zavolá funkciu želanú aplikáciou a vráti jej požadovaný zoznam, čiže aplikácia nezbadala vo výsledku žiadny rozdiel. Popritom však rootkit stihol vykonať aj svoj kód, ktorým napríklad bolo vymazanie informácie o ňom samom v zozname požadovanom aplikáciou. Tým pádom sa rootkitu podarilo zamaskovať svoju existenciu a aplikácia ho neodhalila. Keďže *user mode* rootkity pracujú v jednotlivých pamäťových priestoroch aplikácií, musí takýto typ rootkitu sledovať pamäťové priestory všetkých spustených aplikácií a modifikovať ich činnosti za účelom skrytia svojich činností.

Kernel mode rootkity zahŕňajú skrývanie a modifikácie na úrovni pamäťového priestoru jadra operačného systému. Tento pamäťový priestor je spravidla zakázaný pre neautorizovaných užívateľov a modifikovať ho môže iba užívateľ so zodpovedajúcimi právami. Jadro operačného systému je zároveň najvhodnejším miestom pre systémové skrývanie, pretože je najnižšou vrstvou celého systému a teda umožňuje najspoľahlivejšie možnosti skrývania. Jedným z bežných príkladov použitia *kernel mode* rootkitu je modifikácia dátových štruktúr v pamäti jadra operačného systému. Napríklad, pamäť jadra operačného systému musí udržiavať zoznam všetkých bežiacich procesov a rootkit sa z tohto zoznamu môže jednoducho vymazať rovnako, ako z neho môže vymazať záznamy iných škodlivých softwarov. Tento príklad použitia je známy aj ako priama modifikácia objektu jadra alebo aj DKOM (direct kernel object modification). Iným príkladom použitia takéhoto druhu rootkitu je modifikácia tabuľky SSDT, čo je tabuľka v pamäti jadra operačného systému, ktorá obsahuje adresy funkcií systémových volaní v pamäti jadra. Jednoduchou modifikáciou tejto tabuľky môže rootkit presmerovať vykonávanie na svoj vlastný kód namiesto vykonávania funkcie, ktorá by sa podľa záznamov v tabuľke mala vykonať. Podobne ako bolo uvedené v príklade k *user mode* rootkitu, aj v tomto prípade rootkit zavolá požadovanú funkciu a potom modifikuje jej výstup pred odovzdaním výsledku ďalej tak, aby nebolo možné odhaliť jeho činnosť alebo existenciu.

Spôsobov využitia alebo skôr zneužitia rootkitov ako v *user mode* tak aj v *kernel mode* je samozrejme omnoho viac a vyššie spomenuté slúžia iba ako príklady. Niektoré z týchto spôsobov sú už aj pomerne dobre zdokumentované a využívané pri lepšom návrhu aplikácií, napríklad na detekciu rootkitov. Stále však platí, že rootkity majú schopnosť obchádzať zabezpečenie aplikácií a skrývať seba, či svoju činnosť a preto ich využívanie v škodlivých softwaroch je veľmi časté a rozšírené [4].

2.5 Spyware

Spyware je spravidla druh software, ktorý zbiera osobné informácie bez vedomia užívateľa a odosiela ich tretej strane. Môže sa jednať o relatívne nepodstatné informácie, ako navštívené webové stránky, ktoré tretia strana môže využiť na zasielanie cielenej nevyžiadanej reklamy. Môže ale ísť napríklad aj o dôverné informácie ako sú prihlasovacie údaje, čísla kreditných kariet a podobne, prípadne môže spyware slúžiť aj ako zadné vrátka do užívateľovho systému. Spyware je taktiež schopný

zaznamenávať dianie na pracovnej ploche počítača, zapínať mikrofón, ak je k dispozícii, informovať tretiu stranu o programoch nainštalovaných na počítači alebo dokonca preposielať emailovú komunikáciu z napadnutého počítača.

Toto sú hlavné dôvody, prečo je spyware považovaný za škodlivý software. Jeho sprievodnými znakmi sú problémy s výkonnosťou počítača a s tým súvisiace pomalšie reakcie systému a programov v ňom nainštalovaných, prípadne zvýšená miera reklamy prostredníctvom „vyskakovacích okien“ počas prezerania internetových stránok. Často sa pritom snaží integrovať do internetového prehliadača Internet Explorer, vďaka ktorému bude môcť byť spustený permanentne [5].

2.6 Adware

Adware je druh softwaru zobrazujúci reklamu na počítači. Táto reklama je zobrazovaná vo „vyskakovacích oknách“ a objavuje sa nečakane na pracovnej ploche počítača, aj keď si užívateľ práve neprehliada žiadne internetové stránky. Adware je často sprievodným znakom pri iných softwaroch, ktoré sú užívateľom poskytované zadarmo a táto reklama je v podstate spôsobom zárobku autorov softwaru. Nebezpečenstvo straty alebo odcudzenia informácií pri adware softwari nehrozí, samozrejme ak sa jedná o samotný adware software. Môžu totiž existovať varianty, keď na tento typ softwaru sú naviazané aj iné škodlivé druhy softwaru. Adware je teda skôr software, ktorý užívateľa ruší, obťažuje a okrem toho spomaľuje výkon počítača.

2.7 Vplyv malware na registre

Vplyv vyššie popísaných druhov škodlivého softwaru na registre operačného systému Microsoft Windows je značný, keďže každý z nich nejakým spôsobom zasahuje do registrov a modifikuje ich za účelom dosiahnutia svojich cieľov.

Najbežnejším z množstva zásahov do registrov je modifikácia registrov za účelom zabezpečenia spustenia škodlivého softwaru po naštartovaní počítača resp. systému. Pre tento účel sa používa najčastejšie register *\Software\Microsoft\Windows\CurrentVersion\Run*. Pomerne často sú tiež modifikované napríklad registre *\Software\Microsoft\WindowsNT\CurrentVersion\Winlogon*, za účelom nastavenia lokálneho alebo vzdialeného prístupu, prípadne kľúče registrov, kontrolujúce vykonanie makier v programe Microsoft Word v *Software\Microsoft\Office\11.0\Word\Security\Level* alebo kľúče registrov kontrolujúce doplnky v programe Microsoft Excel v registri *Software\Microsoft\Office\11.0\Excel\Resiliency\StartupItem*. Dva posledné uvedené príklady sa týkajú iba vírusov, kým predchádzajúce sú využívané každým z popisovaných škodlivých softwarov.

Konkrétnejším príkladom zneužitia registrov je trójsky kôň s názvom W32.IRCBot, ktorý s cieľom znemožniť užívateľovi prijímať správy o nakazení od antivírusového programu, modifikoval

register *\SOFTWARE\Microsoft\SecurityCenter\AntiVirusDisableNotify/* a tým vlastne udržiaval užívateľa v stave nevedomosti o nakazení, kým on v systéme vykonával vlastnú škodlivú činnosť [6].

Okrem vyššie spomenutého spôsobu zabezpečenia automatického spustenia škodlivého softwaru po štarte systému sa objavujú aj zložitejšie postupy, ktorými sa škodlivý vírus snaží zamaskovať pred odhalením. Dobrý príkladom je prípad červa W32/Autorun-BBI, ktorý sa najskôr replikuje do zložky systému a následne vytvorí záznam s obsahom *conime.exe* v očakávanom registri *HKLM\Software\Microsoft\Windows\CurrentVersion\Run*. Zaujímavé však je, že červ sa týmto záznamom nepostaral o automatické spustenie seba samého po reštartovaní systému, ale postaral sa o spustenie aplikácie *conime.exe*, ktorá je bežnou súčasťou operačných systémov Microsoft Windows. Trik ale spočíva v nasledujúcej činnosti červa, ktorý vytvorí položku *červ.exe* v registri *HKLM\Software\Microsoft\WindowsNT\CurrentVersion\Image File Execution Options\conime.exe*. Týmto v podstate dochádza k asociácii aplikácie *červ.exe*, ktorou je samotný červ, s aplikáciou *conime.exe*. Vo výsledku to znamená, že červ zabezpečil spustenie aplikácie *conime.exe* automaticky po štarte systému a s touto aplikáciou asocioval sám seba, a tak zabezpečil v podstate nepriamo spustenie samého seba hneď po štarte systému [7].

Popri zneužití štandardnej funkčnosti registrov škodlivým sofwarom popísaným vo vyššie uvedenej časti kapitoly, dochádza často aj k zneužitiu na základe chýb objavených v registroch. Môže sa jednať o chyby, ktoré vznikli v registroch už počas ich vývoja, ale aj o chyby, ktoré do nich vnášajú aplikácie svojimi modifikáciami registrov. Pre lepšiu predstavu si uvedieme niektoré z aktuálne objavených chýb a spôsoby, akým môžu byť zneužité.

Jednou z chýb, ktorá ide na vrub spoločnosti Microsoft, je nedostačujúca kontrola dát registrov, s ktorými pracuje funkcia *RtlQueryRegistryValues()*. Tá pracuje s dátami typu REG_SZ[1], teda reťazec, ktorého dĺžka je však obmedzená. Užívateľ ale vďaka možnosti modifikovať hodnotu registrov s názvom EUDC (End-User-Defined Characters), ktorá určuje kódovanie znakov a umožňuje pracovať so znakmi, ktoré nie sú štandardne dostupné a ktorú vyššie spomenutá funkcia spracováva, môže zadať dĺžku reťazca väčšiu, než funkcia očakáva. V takomto prípade môže dôjsť k prepísaniu pamäte jadra a užívateľ dokáže spúšťať programy so systémovými právami, ku ktorým by za normálnych okolností nemal prístup. Túto chybu využíva trójsky kôň Troj/EUDPoC-A [8].

Ďalšou z chýb, ktorých pôvod leží najpravdepodobnejšie vo vývoji registrov, je chyba operačných systémov Windows Vista a novších, ktoré neprekladajú dôkladne virtuálnu cestu ku kľúču registra na cestu skutočnú. Táto chyba umožňuje užívateľovi spôsobiť reštart systému. Hoci to nie je nebezpečná chyba, mohlo by byť nepríjemné, ak by ju využíval nejaký škodlivý software a systém by sa neustále reštartoval [9].

Jednou z už dlhšie známych chýb, je chyba operačného systému Windows Vista, ktorý nedostatočne chráni informácie o lokálnych užívateľoch v registroch. Toto môže byť zneužité

[1] Typy dát používané v registroch budú bližšie vysvetlené v kapitole 3.7.

lokálnym neprivilegovaným užívateľom k získaniu informácií o ostatných lokálnych užívateľoch, vrátane hesla administrátora uloženého v registroch. Tým možno potenciálne získať kontrolu nad celým systémom [10].

Ako bolo už aj vyššie spomenuté, existujú chyby v registroch, ktoré spôsobí používanie iných aplikácií, ktoré potrebujú do registrov zapísať určité informácie. Ako uvidíme v nižšie uvedených prípadoch, môže sa dokonca stať, že takéto chyby spôsobí software, ktorý by im mal zabraňovať. Preto je problematika registrov naozaj zložitá a z bezpečnostného hľadiska veľmi citlivá.

Napríklad aplikácia pre vzdialené pripojenie k počítaču, No-IP Dynamic Update Client (DUC) 2.2.1, používa nedostatočné práva pre kľúč registra *HKLM\SOFTWARE\Vitalwerks\DUC,* z ktorého možno prostredníctvom tejto chyby získať citlivé informácie, ako napríklad prihlasovacie údaje.

Podobná chyba sa objavila aj v aplikácii VPN-1 SecuRemote/SecureClient, ktorá ukladá informácie o užívateľoch do *HKLM\Software\Checkpoint\SecuRemote,* podkľúča *Credentials,* do ktorého má každý užívateľ plný prístup. Toto umožňuje škodlivému softwaru, ktorému postačujú práva lokálneho užívateľa, získať a zneužiť informácie o všetkých užívateľoch.

Často diskutovanou bola chyba antivírusového programu McAfee VirusScan Enterprise 8.5.0.i, ktorý používal nedostatočné práva pre určité kľúče registrov, ktoré vytvoril. Táto chyba umožňovala lokálnym užívateľom s právom zápisu do registrov, vymazať hodnotu položky UIP z registrov a tým eliminovať ochranu konzoly pre skenovanie vírusov v systéme. Táto hodnota sa spravidla nachádzala v jednom z registrov *HKEY_LOCAL_MACHINE\SOFTWARE\McAfee\DesktopProtection* alebo *\SOFTWARE\NetworkAssociates\TVD\VirusScanEntreprise\CurrentVersion* [11].

Príkladov zneužitia registrov je omnoho viac, ako len vyššie uvedené a preto sú registre primárnym cieľom producentov škodlivých softwarov a útokov z prostredia internetu. Kombinácia minimálnej znalosti problematiky registrov zo strany užívateľov a hlbokých znalostí registrov zo strany útočníkov, dáva útočníkom výrazné možnosti, ako si zabezpečiť získanie potrebných informácií, prostredníctvom škodlivého softwaru. Väčšina softwaru takého druhu, aký bol popísaný vyššie, aktívne pracuje s registrami, čo mu umožňuje skrývať sa a dokonca v prípade vymazania, mu umožňuje opätovnú reinštaláciu. Tieto fakty dokazujú potrebu venovať pozornosť rozvoju softwaru pre zabezpečenie registrov operačných systémov Windows, ku ktorým patria hlavne antivírusové programy.

3 Technológie uloženia dát aplikácií a operačných systémov

Vzhľadom na zameranie tejto práce, považujem za dôležité uviesť technológie a spôsoby ukladania dát aplikácií i operačných systémov ako aj ich postupný vývoj. Je však potrebné upozorniť, že nejde o ukladanie ľubovoľných dát, ale o ukladanie konfiguračných dát aplikácií a operačných systémov.

3.1 Uloženia dát operačných systémov

V tejto časti sa budeme venovať iba dvom najrozšírenejším druhom operačných systémov, a to systémom Microsoft Windows a systémom založeným na Unixe. Rozdiely medzi uložením konfiguračných dát týchto operačných systémov sa postupom času menili, a hoci sa dnes odlišujú, nebolo tomu vždy tak.

Tak ako väčšinu oblastí informačných technológií, aj oblasť ukladania konfiguračných dát operačných systémov Microsoft Windows bola ovplyvnená technologickým vývojom. Kým prvá verzia operačného systému pod názvom Windows 3.1 sa objavila v časoch, keď priemerná kapacita pevných diskov dosahovala približne 80 MB, o dva roky neskôr, v roku 1994 bola ich priemerná veľkosť už 400 MB a rýchlo sa zvyšovala. Toto malo priamy dosah aj na ukladanie konfiguračných dát.

Pri používaní Windows 3.1 sa predpokladalo použitie minima aplikácií a teda aj minima nastavení, ktoré s danými aplikáciami a samotným systémom súviseli. Preto bol zavedený jednoduchý spôsob vytvárania súborov s koncovkou .INI. Tento spôsob počítal s tým, že každá aplikácia a aj samotný operačný systém bude mať svoj vlastný súbor *.ini*, v ktorom bude mať uložené konfiguračné dáta. Samotné operačné systémy Windows 3.x využívali dva typy takýchto súborov. Prvým boli systémové inicializačné súbory, medzi ktoré patrili súbor *SYSTEM.INI*, ktorý obsahoval informácie o hardwari počítača, a súbor *WIN.INI*. V ňom boli združené informácie o užívateľských plochách, užívateľských nastaveniach a aplikáciách systému. Druhým typom boli súkromné inicializačné súbory, ku ktorým patrili súbory *CONTROL.INI*, *PROGRAM.INI*, *WINFILE.INI* a *PROTOCOLS.INI* ako aj *.ini* súbory jednotlivých aplikácií.

S príchodom systému Windows 3.11 bol okrem *.ini* súborov zavedený aj ďalší typ súborov pre ukladanie konfiguračných dát a boli nimi binárne súbory s koncovkou *.dat*. Konkrétne v systéme Windows 3.x sa nachádzal súbor *REG.DAT*, ktorý obsahoval asociácie typov súborov k jednotlivým programom. Pre samotné spustenie operačného systému však boli najdôležitejšie iné súbory, než súbory *.ini* a *.dat*. Išlo o súbor *CONFIG.SYS*, ktorého príkazy sa vykonávali pri štarte systému a zabezpečovali funkčnosť hardwarových prvkov, a o súbor *AUTOEXEC.BAT*, ktorý sa automaticky

spúšťal pri štarte systému a zabezpečoval spustenie rezidentných programov a nastavenie predvoleného stavu operačného systému. A hoci bez týchto dvoch súborov, ktoré obsahovali dôležité konfiguračné dáta systému, by systém nebolo možné spustiť, najdôležitejšou úlohu pri vývoji uloženia dát v rámci systému zohrávali súbory s koncovkou *.ini*. Týchto súborov bolo totiž najviac a boli s nimi spojené problémy, ktoré výrazne ovplyvnili ďalší vývoj v oblasti uloženia konfiguračných dát operačných systémov.

Problémom napríklad bolo, že nebolo definované, kde majú byť tieto súbory uložené. Preto došlo k situácii, že si ich každá aplikácia ukladala kam chcela a nakoniec boli roztrúsené po celom pevnom disku. Toto malo samozrejme za následok, že ich nebolo možné efektívne spravovať a optimalizovať. Ďalším problémom bolo neustále zväčšovanie priemernej kapacity disku, s čím logicky súvisela možnosť používania viacerých aplikácií, čo by spôsobilo ešte väčší chaos. Preto bola s príchodom verzie Windows 95 uvedená aj nová technológia ukladania konfiguračných dát, a to pomocou registrov.

Registre sú v podstate databázou slúžiacou pre efektívnejšie uloženie a vyhľadávanie jednotlivých konfiguračných dát a nastavení. Ich zavedenie bolo prijaté pozitívne, nakoľko ponúkali ucelený prostriedok prístupu k týmto dátam. Prvotná verzia registrov, ktorá bola v porovnaní s dnešnou verziou neopracovaná, prešla čoskoro úpravami pre verziu operačného systému Windows 3.5 NT. Táto verzia systému používala pre ukladanie obsahu registrov súbory s koncovkou *.dat*. Najdôležitejšími boli súbor *SYSTEM.DAT*, pre ukladanie systémových nastavení a informácií o hardwari, a súbor *USER.DAT*, pre ukladanie užívateľských nastavení.

K ďalšej zmene v súvislosti so zavedením registrov došlo vo verzii Windows 95, v ktorej už pre štart systému nebolo potrebné používať súbory *CONFIG.SYS* a *AUTOEXEC.BAT*, nakoľko ich funkciu prebrali registre a súbor *IO.SYS*. V ďalších verziách operačných systémov, ktorými boli Windows 98 a Windows Me, sa však už registre zmien nedočkali, čo vyvolalo vývoj softwaru pre čistenie a opravu registrov. Tomuto druhu softwaru sa darilo a prosperoval až kým nebola na trh uvedená verzia operačného systému Windows 2000 a následne Windows XP a Windows Vista. V týchto verziách totiž došlo k transformácii registrov s cieľom vyhovieť náročnejším sieťovým požiadavkám a pre uloženie registrov sa s výnimkou súboru *NTUSER.DAT* začali používať binárne súbory bez koncovky s názvami *Default*, *SAM*, *Security*, *Software* a *System*. Tieto transformácie vykonané v registroch boli zásadnejšieho rázu a tak už prevažná väčšina aplikácií pre opravu a čistenie registrov nefungovala správne [12].

Budúcnosť môže z pohľadu registrov a ukladania konfiguračných dát operačných systémov Microsoft Windows priniesť zásadné zmeny, možno ich však očakávať až v dlhodobom horizonte a to minimálne z dôvodu kompatibility [13].

Uloženie konfiguračných dát operačných systémov založených na Unixe sa od uloženia v operačných systémoch Windows zásadne líši. Konfiguračné dáta samotného operačného systému sú spravidla uložené v adresári */etc*. Tento adresár obsahuje prípadne ďalšie adresáre a konfiguračné

súbory v textovej forme, ku ktorým má však právo čítania a editácie iba správca systému. V prípade získania týchto práv útočníkom, by bol celý systém ohrozený. Okrem toho platí, že ak dôjde k poškodeniu niektorého z konfiguračných súborov, služba ku ktorej patril, prestane fungovať. Je však veľmi nepravdepodobné, že by k takémuto poškodeniu došlo, keďže súbory v tomto adresári sú modifikované len zriedka, má k nim prístup iba správca systému a keďže sú v textovej forme, sú aj relatívne ľahko opraviteľné. Tieto skutočnosti pridávajú unixovým systémom na stabilite, v porovnaní s operačnými systémami Windows, kde v prípade poškodenia registrov, ktoré sú najčastejšie modifikovaným súborom systému, dochádza k znefunkčneniu celého systému. Samozrejme existujú služby, ktorých konfiguračné dáta a nastavenia musí mať možnosť modifikovať aj bežný užívateľ. V takomto prípade existuje okrem súboru v adresári */etc* aj súbor v domovskom adresári daného užívateľa odlíšený bodkou na začiatku názvu, ktorý môže modifikovať aj bežný užívateľ. Ako príklad možno uviesť aplikáciu wgetc, ktorej jeden konfiguračný súbor s názvom *wgetrc* je uložený v adresári */etc* a druhý s názvom *.wgetrc* je uložený v domovskom adresári užívateľa.

Jedným z najdôležitejších adresárov v adresári */etc* je adresár s názvom *init.d*, ktorý obsahuje množstvo skriptov zabezpečujúcich spustenie služieb a procesov potrebných pre správny beh systému. Spúšťanie týchto procesov a služieb však nie je náhodné a je riadené ďalšími konfiguračnými súbormi. Pre operačný systém Linux Gentoo napríklad platí, že sa najskôr spúšťajú skripty, na ktoré existuje odkaz z konfiguračného súboru */etc/runlevels/boot* a následne tie, na ktoré existuje symbolický odkaz z konfiguračného súboru */etc/runlevels/default*. Skripty sa spravidla spúšťajú v abecednom poradí až na výnimky, keď je toto poradie porušené z dôvodu závislostí procesov na iných procesoch. Dôležitým konfiguračným súborom pri spúšťaní skriptov z adresára *init.d* je aj súbor */etc/inittab*, ktorý určuje v akom režime budú služby a procesy z adresára *init.d* spustené, a tým vymedzuje správanie sa systému pri štarte. Hoci spomínané konfiguračné súbory *boot* a *default* sa nenachádzajú v každej distribúcii Linuxu, popísaný princíp spúšťania skriptov z adresára *init.d* je platný vo väčšine aktuálnych distribúcii unixových systémov [15].

Vo všeobecnosti však nemožno vymedziť typy konfiguračných súborov a spôsob ukladania konfiguračných dát v operačných systémoch založených na Unixe. Špecifikom týchto systémov je práve voľnosť, ktorá umožňuje využitie rôznych spôsobov uloženia nastavení a konfiguračných dát od jednoduchých súborov obsahujúcich zoznamy nastavení alebo príkazov až po komplexné súbory s koncovkou .CONF. Pre samotné unixové operačné systémy však platí, že najdôležitejšie konfiguračné dáta sa nachádzajú v už spomenutom adresári */etc*, ktorý obsahuje aj niekoľko súborov s koncovkou .CONF. Ako príklad možno uviesť súbory *host.conf*, *inetd.conf* a ďalšie.

Rozdiely medzi unixovými operačnými systémami a systémami Microsoft Windows v náväznosti na konfiguračné dáta možno badať aj v práci s úložiskom konfiguračných dát jednotlivých systémov. Kým unixové systémy pracujú so súbormi v adresári */etc* iba keď potrebujú informácie v nich uložené, Windows musí už aj pred spustením aplikácie skenovať registre

a zisťovať, či neobsahujú určité konfiguračné nastavenia. Záver je taký, že Windows trávi oveľa viac času činnosťou spojenou s registrami, než unixové systémy činnosťou spojenou so súbormi v adresári */etc*. A hoci sa možno na začiatku javila technológia ukladania konfiguračných dát operačného systému do jednoduchých súborov ako neefektívna v porovnaní s registrami, vyššie uvedené informácie dokazujú, že táto domnienka bola mylná.

3.2 XML

XML je značkovací jazyk slúžiaci predovšetkým pre výmenu dát medzi aplikáciami a pre publikovanie dokumentov, pričom popisuje ich štruktúru a význam. Výhodou formátu XML je, že je založený na obyčajnom texte a teda aj keď pre väčšinu ľudí zostáva kód XML skrytý a budú ho používať iba aplikácie pre vzájomnú komunikáciu, nie je problémom XML dokument otvoriť a modifikovať v ľubovoľnom textovom editore. Ďalším typickým znakom XML je možnosť vytvárania vlastných značiek s definovaným významom.

Toto nachádza okrem iného veľké uplatnenie pri ukladaní dát aplikácií, ktoré potrebujú pre správnu funkčnosť určité konfiguračné dáta. Tie síce možno ukladať pre opätovné použitie aj do textových súborov alebo registrov, tieto spôsoby sú však pri zložitejších konfiguračných dátach zbytočne komplikované a často v praxi aj nevyužiteľné, hlavne v prípade textových súborov. Preto sa použitie XML s prípadnými vlastnými značkami javí ako optimálne riešenie a navyše užívateľ so znalosťou problematiky by mal možnosť tento súbor modifikovať aj bez použitia samotnej aplikácie.

V súčasnosti patrí XML k najrozšírenejším formátom ukladania konfiguračných dát aplikácií. Napomáha tomu aj skutočnosť, že sa vytvárajú štandardy, vďaka ktorým je tieto dáta možné prenášať medzi aplikáciami bez straty informácií. Ich spracovanie jednotlivými aplikáciami je potom už len vecou použitia niektorej z existujúcich knižníc implementujúcich spracovanie súborov vo formáte XML [14].

3.3 YAML

YAML je jazyk, ktorý podobne ako jazyk XML umožňuje uloženie konfiguračných dát aplikácie a ich výmenu medzi aplikáciami. Je medzi nimi však zásadný rozdiel.

Hoci sa zdalo, že jazyk XML bude pre ukladanie dát aplikácií dominovať, časom sa zistilo, že pre niektoré prípady je príliš mohutný. Skutočnosť, že veľké množstvo aplikácií nepotrebuje využívať všetky prvky jazyka XML, ako napríklad šablóny, menné priestory alebo DTD, bola impulzom pre vývoj jednoduchšieho jazyka, ktorým je práve jazyk YAML. Tento jazyk je veľmi jednoduchý, bez zložitých konštrukcií, ale zároveň je schopný vyjadriť v čistom texte aj zložitejšie konštrukcie, akými sú napríklad polia či štruktúry. Okrem toho dokáže zapisovať štruktúrované dáta podobne ako jazyk XML, avšak s menšou réžiou a s vyššou užívateľskou prívetivosťou. YAML dokonca dokáže vkladať

binárne dáta alebo spájať informácie z rôznych dokumentov. Jeho ďalšími vlastnosťami, ktoré hovoria o jeho značnom potenciály pri ukladaní konfiguračných dát aplikácií, sú napríklad jednoduché použitie a implementácia, jednoduchá možnosť čítania a úprav užívateľom a značné rozšírenie knižníc implementujúcich prácu so súbormi vo formáte YAML [16].

3.4 Súbory .properties

Súbory s príponou *.properties* sú súbormi pre uloženie konfiguračných dát aplikácií v programovacom jazyku Java. Hoci v tomto jazyku, podobne ako aj v mnohých iných, nachádzajú široké uplatnenie súbory vo formáte XML, ktorý je popísaný v kapitole 3.2, svoju dôležitú pozíciu majú aj súbory *.properties*, ktoré sa od XML líšia.

XML umožňuje ukladať komplexnejšiu reprezentáciu konfiguračných dát, zložitejšie štruktúry a vzťahy a väčšinou sa používa vtedy, keď je zrejmé, že aplikácia bude musieť ukladať väčšie množstvo konfiguračných dát. Pre menšie množstvá dát, prípadne pre jednoduchšie dáta, sú plne postačujúce súbory *.properties*, ktorých vyjadrovacia schopnosť je síce oproti XML značne obmedzená, pre jednoduché aplikácie však plne postačujúca. Výhodou týchto súborov je prehľadnosť, keďže konfiguračné dáta sú spravidla uložené v pároch „kľúč hodnota“ pričom každý záznam je uložený na novom riadku súboru.

3.5 Databázy

Databázy patria taktiež k jedným z mnohých spôsobov uloženia konfiguračných dát aplikácií. Využitie nachádzajú predovšetkým v dlhodobo bežiacich a väčších webových aplikáciách a webových službách.

Uloženie dát užívateľského profilu a nastavení do databázy, napríklad pri webových službách, je o mnoho jednoduchšie, pretože centralizované úložisko dát umožňuje hocikedy tieto nastavenia a dáta modifikovať a okamžite ich používať. Pri použití súboru pre uloženie konfiguračných dát by vznikala zbytočná réžia spojená s transportom tohto súboru k poskytovateľovi služby a zrejme aj časové oneskorenia spojené so zavedením zmien do nastavení poskytovanej služby. Ak by potom takýto systém poskytujúci určitú službu mal veľké množstvo užívateľov, možno predpokladať, že by väčšinu času strávil prácou s týmito súbormi namiesto toho aby bol výkon systému využitý na poskytovanie služby. Preto je využívanie databáz v oblasti webových služieb veľmi logickým a rozumným riešením. Databáz, ktoré je takto možné využiť v spojení s určitými službami je veľké množstvo, pričom medzi najznámejšie patria MySQL a PostgreSQL. Ich prepojenie so servermi, ktoré poskytujú dané služby, taktiež nie je problematické, nakoľko väčšinu mail serverov, FTP serverov, či web serverov je možné prepojiť prakticky s každým databázovým systémom.

Okrem vyššie spomenutých faktov je veľkou výhodou databáz ich jednoduchá údržba, keďže všetky konfiguračné dáta sú uložené v jednej databáze a nie je potrebné vyhľadávať konkrétny konfiguračný súbor s príslušnými dátami. Táto skutočnosť umožňuje zdieľanie konfiguračných dát medzi väčším počtom počítačov bez potreby kopírovania jednotlivých konfiguračných súborov. K ďalším výhodám patrí možnosť vytvárania logov, čiže záznamov o zmenách, prípadne vytváranie záloh a s tým súvisiaca možnosť obnovenia databáz pri poškodení. Tieto výhody neposkytuje žiadna iná technológia uloženia konfiguračných dát, a preto sú databázy pomerne často vyžívané pre ukladanie týchto dát.

3.6 INI súbory

Súbory s príponou *.ini*, ako bolo spomenuté už v kapitole 3.1, boli súbory pre uloženie konfiguračných dát prvých verzií operačných systémov Microsoft Windows.

Štruktúra týchto súborov je jednoduchá a skladá sa len z dvoch úrovní, ktorými sú sekcie a reťazce, ktoré patria pod určitú sekciu a obsahujú vlastné konfiguračné dáta. Súbory *.ini* sú samozrejme čitateľné v ľubovoľnom textovom editore, keďže ide o textové súbory. Ich problémom však bola ich veľkosť obmedzená na 64 KB, čo je pri väčšom množstve konfiguračných dát nedostačujúca veľkosť a pri ukladaní konfiguračných dát operačného systému, bol predpoklad vzniku veľkého množstva takýchto dát samozrejmý. Ďalšou nepríjemnou skutočnosťou je možnosť otvorenia súboru *.ini* aplikáciou v exkluzívnom móde, ktorý zamkne súbor z pohľadu iných aplikácií. Ak potom takýto súbor, ktorý je zdieľaný viacerými aplikáciami, obsahuje dôležité informácie, napríklad o bezpečnostných právach, iné aplikácie tieto práva nie sú schopné zistiť, čo môže zamedziť ich funkčnosti. Ak naopak pristupujú dve aplikácie k rovnakému *.ini* súboru naraz a obe ho modifikujú, neskôr vykonaná modifikácia odstráni zmeny vykonané skoršou modifikáciou. Tieto dôvody, rovnako ako aj dôvody uvedené v kapitole 3.1, vyústili k obmedzeniu používania súborov *.ini* pre uloženie konfiguračných dát operačného systému a podnietili vznik tzv. registrov.

Hoci v súčasnosti spoločnosť Microsoft odporúča namiesto používania súborov *.ini* používať registre, existuje množstvo aplikácií, ktoré pre ukladania svojich konfiguračných dát naďalej využívajú tento druh súborov. Hlavným dôvodom je, že v dnešnej dobe sú už registre príliš zložité a pre aplikácie, ktoré nepotrebujú ukladať veľké množstvo konfiguračných dát, je spôsob ukladania a prístupu do *.ini* súborov omnoho rýchlejší a efektívnejší. Tým, že každá z aplikácií využíva iba svoj *.ini* súbor, sa eliminovali vyššie popisované problémy a teda možno predpokladať, že využívanie takéhoto druhu súborov pre ukladanie konfiguračných dát aplikácií v operačnom systéme Microsoft Windows bude pretrvávať. V každom prípade však jednoznačný prechod od *.ini* súborov k registrom v prípade operačných systémov je nespochybniteľný [17].

3.7 Registre

Registre sú hierarchickou databázou, ktorá sa používa v operačných systémoch Microsoft Windows, od verzie Windows 95 a slúži k ukladaniu konfiguračných dát a informácií operačného systému, aplikácií a hardwarových zariadení.

Na jednej strane sú registre iba mohutným úložiskom dát a informácií, ktoré je pre bežného užívateľa zdanlivo nepodstatné, keďže s ním do priameho styku nepríde. Na druhej strane však registre zohrávajú kľúčovú úlohu pri fungovaní a správaní sa celého systému. Registre totiž obsahujú informácie, ktoré operačný systém Windows používa neustále počas svojho behu a ktoré sú nenahraditeľné pre vykonávanie rôznych operácií a beh aplikácií. Medzi tieto informácie patria napríklad profily jednotlivých užívateľov, aplikácie nainštalované v počítači, typy dokumentov, ktoré môžu jednotlivé aplikácie vytvárať, nastavenia stránok, vlastností zložky, ikony aplikácií, informácie o hardware existujúcom v počítači a o používaných portoch a mnoho ďalších.

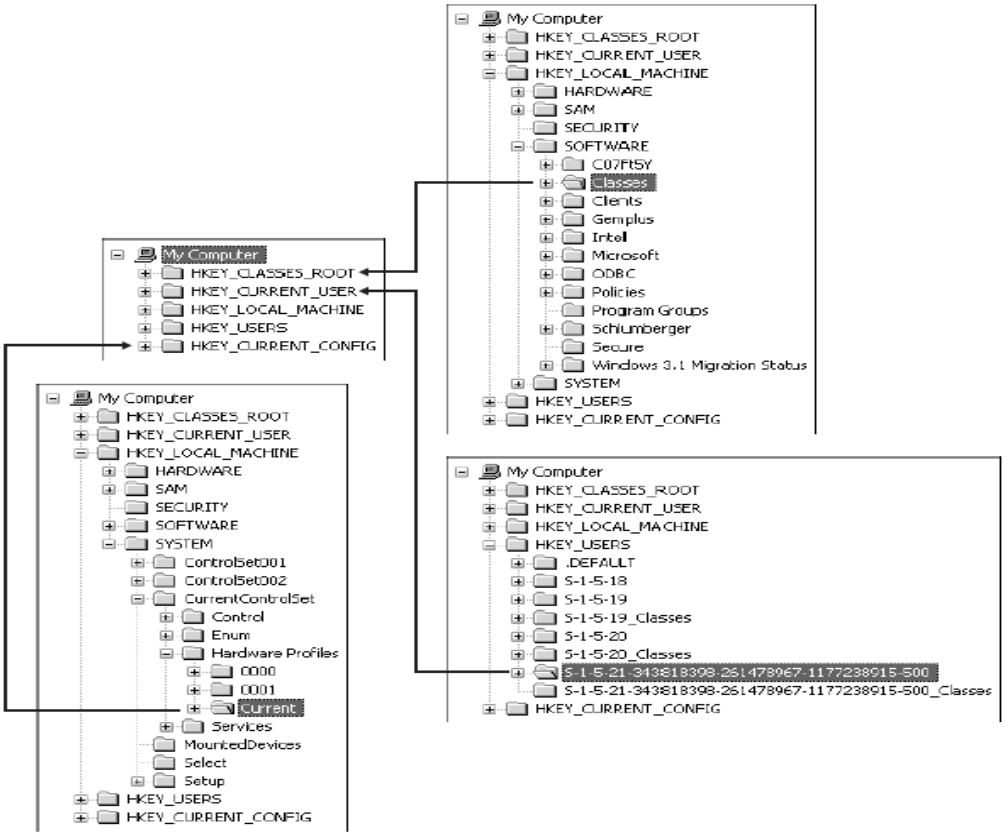
Štruktúra registrov operačných systémov Windows sa podobá štruktúre súborového systému Windows. Tak ako možno v súborovom systéme sledovať hierarchiu súborov a priečinkov, ktorej koreňom je „Tento počítač“ a hlavnými položkami sú pevný disk, CD mechanika a ďalšie, a v hlavných položkách sa nachádza množstvo ďalších dát, priečinkov a súborov, možno takúto hierarchiu sledovať aj pri registroch. Pomocou nástroja regedit, ktorý bude popísaný v kapitole 3.8.1, možno vidieť, že koreňom v štruktúre registrov je opäť položka „Tento počítač“ v ktorej sa nachádzajú hlavné položky registrov a v nich sa nachádza ďalšie množstvo dát. Túto štruktúru si popíšeme podrobnejšie.

Tabuľka 3.1: Koreňové kľúče registrov

Názov koreňového kľúča	Používaná skratka	Popis
HKEY_CLASSES_ROOT	HKCR	Obsahuje dva typy nastavení. Prvým je priradenie súborov, pri ktorom sa rôzne typy súborov priraďujú k aplikáciám a programom, ktoré ich otvárajú, tlačia a upravujú. Druhým typom je registrácia tried pre objekty COM (Component Object Model) .Tento koreňový kľúč je najväčšou položkou registrov a umožňuje meniť chovanie operačného systému.
HKEY_CURRENT_USER	HKCU	Obsahuje koreňovú zložku informácií o konfigurácii pre aktuálne prihláseného užívateľa. Sú tu uložené zložky užívateľa, farby obrazovky, sieťové pripojenia, predvoľby aplikácií a nastavenia Ovládacích panelov.
HKEY_LOCAL_MACHINE	HKLM	Obsahuje informácie o konfigurácii daného počítače (platia pre každého užívateľa)
HKEY_USERS	HKU	Obsahuje všetky aktívne načítané užívateľské profily v počítači. Kľúč HKEY_ CURRENT_USER je podkľúčom tohto kľúča.
HKEY_CURRENT_CONFIG	HKCC	Obsahuje informácie o hardwarovom profile používanom počítačom pri spustení systému.

Hlavné položky nachádzajúce sa pod koreňovou položkou sa odborne nazývajú koreňové kľúče (root keys) a ich výpis aj so stručným popisom sa nachádza v tabuľke 3.1. V koreňových kľúčoch sa nachádzajú ďalšie kľúče, ktoré sa ďalej rôzne členia a zanorujú. Tieto kľúče sa potom nazývajú podkľúčmi. Samotné kľúče sa podobajú priečinkom alebo zložkám v súborovom systéme. Hlavnú podobnosť možno badať v pomenovávaní kľúčov a ich umiestnení. Možno totiž vložiť ľubovoľné množstvo kľúčov do iného kľúča, ak názvy v danom kľúči zostanú jedinečné.

Najdôležitejšie z týchto koreňových kľúčov sú kľúče HKLM a HKU. Iba tieto dva koreňové kľúče totiž ukladá systém Windows skutočne na pevný disk. Ostatné kľúče predstavujú iba prepojenia k podkľúčom v HKLM alebo HKU, ako je to znázornené na obrázku 3.1.



Obrázok 3.1: Prepojenia k podkľúčom v HKLM a HKU.

Z pohľadu prepojenia koreňových kľúčov k podkľúčom, došlo vo verziách registrov používaných v aktuálnych operačných systémoch k dôležitej zmene. Táto zmena sa týka koreňového kľúča HKCR. Tento koreňový kľúč už nie je tvorený iba podkľúčom *HKLM\SOFTWARE\Classes*, ako to znázorňuje obrázok 3.1, ale od verzie operačného systému Windows 2000, je zložitejší. Pre jeho odvodenie zlučuje operačný systém kľúč *HKLM\SOFTWARE\Classes*, ktorý obsahuje predvolené priradenia súborov a registrácie tried, čiže platné pre celý systém, a kľúč *HKCU\SOFTWARE\Classes*, ktorý obsahuje užívateľské priradenia súborov a registrácie tried.

Koreňové kľúče, ako už bolo spomenuté, môžu obsahovať ľubovoľné množstvo ďalších kľúčov. Tieto kľúče opäť môžu obsahovať ľubovoľné množstvo kľúčov a tak dochádza k prehlbovaniu hierarchie jednotlivých kľúčov. Okrem ďalších kľúčov však kľúče obsahujú aj jednu alebo viac hodnôt. Pre porovnanie so súborovým systémom si možno predstaviť kľúč ako priečinok a hodnotu ako súbor v tomto priečinku. Každá hodnota má tri atribúty a to názov, typ hodnoty a dáta.

Každá hodnota musí mať názov. Ak tento názov nie je zadaný, na jeho mieste sa v nástroji regedit uvádza označenie (Default). Platí však, že v každom kľúči musia byť názvy hodnôt jedinečné, odlišné kľúče však môžu obsahovať hodnoty s rovnakými názvami.

Dáta hodnoty môžu byť prázdne, nulové alebo môžu obsahovať konkrétne dáta. Ich veľkosť je maximálne 32 767 bajtov, skutočný limit však je 2 kilobajty. Dáta hodnoty sú spravidla typu zodpovedajúcemu typu hodnoty, výnimkou môžu byť iba binárne dáta, ktoré môžu obsahovať reťazce, štvorbajtové čísla alebo hocičo iné.

Typ hodnoty, ktorý musí byť pri každej hodnote určený, obsahuje informáciu o type dát hodnoty. Prehľad týchto typov aj s ich popisom obsahuje tabuľka 3.2.

Tabuľka 3.2: Typy hodnôt registrov

Typ hodnoty	Typ dát	Popis
REG_NONE	Nie je k dispozícii	Hodnota bez definovaného typu. Takéto dáta sú do registrov zapísané systémom alebo aplikáciou a sú zobrazené nástrojom regedit v šestnástkovom formáte ako binárne dáta.
REG_BINARY	Binárna hodnota	Sú zobrazované v šestnástkovom formáte, v ktorom ich je potrebné aj zapisovať. Väčšina informácií o hardware sa zapisuje pomocou binárnych dát.
REG_DWORD	4-bajtové číselné hodnoty	Zobrazujú a upravujú sa v desiatkovom alebo šestnástkovom formáte a slúžia pre uloženie booleovských príznakov alebo času v milisekundách.
REG_ DWORD_BIG_ENDIAN	4-bajtové číselné hodnoty	Podobné ako REG_DWORD stým rozdielom bajty sa do pamäte ukladajú v opačnom poradí ako pri REG_DWORD.
REG_ DWORD_LITTLE_ENDIAN	4-bajtové číselné hodnoty	Formát totožný s REG_DWORD.
REG_QWORD	8-bajtové číselné hodnoty	Podobné typu REG_DWORD s tým rozdielom, že ide o 8-bajtové číslo. Tento formát je podporovaný výlučne systémom Windows XP Professional x64 Edition.
REG_ QWORD_LITTLE_ENDIAN	8-bajtové číselné hodnoty	Formát totožný s REG_QWORD.

REG_ QWORD_BIG_ENDIAN	8-bajtové číselné hodnoty	Podobné ako REG_QWORD stým rozdielom bajty sa do pamäte ukladajú v opačnom poradí ako pri REG_QWORD.
REG_SZ	Reťazcová hodnota	Textový reťazec s pevnou dĺžkou. Spolu s REG_DWORD najčastejšie používaný typ.
REG_EXPAND_SZ	Rozšíriteľná reťazcová hodnota	Textový reťazec s premennou dĺžkou. Môže obsahovať premenné prostredia, ktoré sú použité pri spustení programu.
REG_MULTI_SZ	Viacreťazcová hodnota	Binárne hodnoty obsahujúce zoznam reťazcov, kde reťazce sú oddelené nulovými znakmi (0x00) a zoznam je ukončený 2 nulovými znakmi.
REG_RESOURCE_LIST	Binárna hodnota	Zoznam hodnôt REG_FULL_RESOURCE_DESCRIPTOR. Nástroj Regedit umožňuje tieto hodnoty zobrazovať ale nie modifikovať.
REG_RESOURCE_REQUIREMENTS_LIST	Binárna hodnota	Zoznam možných hardwarových prostriedkov ovládača zariadenia, ktoré môže tento ovládač alebo jedno s im ovládaných fyzických zariadení používať.
REG_FULL_RESOURCE_DESCRIPTOR	Binárna hodnota	Zoznam prostriedkov používaných fyzickým hardwarovým zariadením.
REG_LINK	Odkaz	Prepojenie. Reťazec znakov udávajúci názov symbolického odkazu.

Spojenie hodnôt, ich názvov, typov a dát s kľúčmi, podkľúčmi a koreňovými kľúčmi a ich vzájomné prepojenia, vytvára logickú štruktúru registrov. Logická štruktúra však nehovorí nič o tom, aké je skutočné usporiadanie dát registrov na pevnom disku.

Fyzicky systém usporadúva registre do podregistrov, z ktorých každý je uložený v binárnom súbore označovanom ako súbor podregistra. Pre každý zo súborov podregistrov vytvára systém Windows ďalšie podporné súbory obsahujúce záložné kópie všetkých dát podregistrov. Tieto záložné súbory umožňujú operačnému systému opraviť podregister behom procesu inštalácie alebo spúšťania systému, ak došlo k nejakej závažnej chybe. Podregistre existujú iba pre koreňové kľúče HKLM a HKU. Ich štandardným úložiskom je v prípade súborov podregistrov HKLM priečinok *%SystemRoot%\System32\config* a v prípade súborov podregistrov HKU priečinky jednotlivých užívateľských profilov.

Pre koreňový kľúč HKU platí, že každý jeho podkľúč je podregistrom. Napríklad podkľúč *HKU\.DEFAULT* je podregistrom a súbor, v ktorom je uložený sa nachádza v priečinku *%SystemRoot%\System32\config*. Zakaždým, keď sa k systému Windows prihlási nový užívateľ, použije operačný systém tento podregister k vytvoreniu nového profilu pre daného užívateľa. Jeho

profil bude následne uložený do súboru *ntuser.dat*, čo je podregister daného užívateľského profilu. Tento súbor sa potom bude nachádzať v domovskom priečinku daného užívateľa.

V prípade podregistrov koreňového kľúča HKLM je situácia odlišná a neplatí tu, že každý podkľúč je podregistrom. Prehľad súborov podregistrov koreňového kľúča HKLM a súborov k nim podporných je uvedený v tabuľke 3.3.

Tabuľka 3.3: Súbory podregistrov

Podregister	Súbor podregistra	Podporné súbory
HKLM\SAM	sam	sam.log
HKLM\SECURITY	security	security.log
HKLM\SOFTWARE	software	software.log, software.sav
HKLM\SYSTEM	system	system.log, system.sav

Z tabuľky 3.3 je zrejmé, že jediný z podkľúčov HKLM, ktorý nemá súbor podregistra, je podkľúč *HKLM\HARDWARE*. Je to spôsobené tým, že tento podregister je dynamický a Windows ho vytvára pri každom zavádzaní operačného systému opätovne a pri vypnutí neukladá podregister ako súbor podregistra.

Z hľadiska fyzického uloženia registrov systému Windows je potrebné poznamenať, že veľkosť súborov podregistrov je prakticky neobmedzená. V systéme Windows 2000 ešte obmedzená bola, avšak zmenou spôsobu mapovania registrov do pamäte sa podarilo vo verziách Windows XP a vyšších, tomuto neželanému obmedzeniu zabrániť.

Z vyššie uvedených informácií je zrejmé, že štruktúra registrov nie je triviálna, takisto ako nie je jednoduchý ani ich spôsob uloženia, keďže fyzicky nie sú uložené v jednom súbore. V porovnaní s ukladaním do konfiguračných súborov *.ini*, však majú registre značné výhody. Patrí medzi ne napríklad jednoduchá možnosť zálohy konfiguračných dát, ktorá by pri používaní necentralizovanej metódy akou boli napríklad *.ini* súbory, bola značne náročnejšia. Okrem toho, vďaka tejto centralizovanej metóde prístupu ku všetkým konfiguračným dátam, je vždy zrejmé kde je potrebné pátrať po nastaveniach operačného systému alebo aplikácií, keďže sú všetky uložené v registroch. Toto zároveň umožňuje prispôsobovanie si funkčnosti, vzhľadu či iných nastavení operačného systému a aplikácií, ktoré nie je možné vykonať žiadnym iným spôsobom [18][19].

3.8 Aktuálne spôsoby prístupu k registrom

Z predchádzajúcich kapitol vyplýva potreba upriamenia pozornosti na registre operačného systému Microsoft Windows, ktoré sú častým útočiskom škodlivého softwaru. Z kapitoly 3.7 je však zrejmé, že spôsob uloženia registrov nie je vôbec jednoduchý a preto možno predpokladať, že práca s nimi a prístup k nim, taktiež nebude jednoduchou vecou.

Nasledujúca kapitola sa venuje stručnému popisu najrozšírenejších nástrojov a metód prístupu k registrom a práce s nimi.

3.8.1 Regedit

Regedit je systémový nástroj operačných systémov Microsoft Windows pre prístup a prácu s registrami, ktorý je ich súčasťou od počiatku používania registrov pre uloženie konfiguračných dát operačného systému.

V starších verziách systému Windows existovali, na rozdiel od aktuálnych verzií, dva nástroje pre prácu s registrami. Prednosťou prvého s názvom Regedit.exe, bolo vyhľadávanie, vďaka ktorému bolo možné v registroch jednoduchšie a ľahšie vykonávať zmeny. Problémom však bolo, že nebolo možné vykonávať ľubovoľné zmeny. Napríklad nebolo možné zobrazovať ani modifikovať hodnoty typov REG_EXPAND_SZ a REG_MULTI_SZ. Pri pokuse o ich zobrazenie dôjde k zobrazeniu vo forme binárnych dát a pri pokuse o ich modifikáciu, budú síce hodnoty týchto dátových typov zmenené, avšak modifikované hodnoty budú uložené ako typ REG_SZ, ktorý už nemusí spĺňať zamýšľanú funkciu. Preto ak chcel užívateľ použiť niektoré spomenuté funkcie, bolo pre neho vhodnejšie použiť druhý zo spomínaných nástrojov Regedt32.exe, aby sa tým vyhol prípadnému fatálnemu poškodeniu registrov spôsobených nekorektnou prácou s dátovými typmi nástrojom Regedit.exe.

Nástroj Regedt32.exe dokáže s dátovými typmi používanými v registroch pracovať korektne. Jeho ďalšou veľkou výhodou bola schopnosť nastavovania zabezpečenia jednotlivých kľúčov registrov a tiež podpora možnosti exportu konkrétnych kľúčov do súborov podregistrov a ich obnova z týchto súborov.

Postupným vývojom však došlo k spojeniu výhod oboch nástrojov do jedného a hoci v operačných systémoch stále existujú oba nástroje, nástroj Regedt32.exe dnes už len spúšťa samotný nástroj Regedit.exe. K tomuto zlúčeniu došlo vo verzii Microsoft Windows XP a trvá dodnes [20].

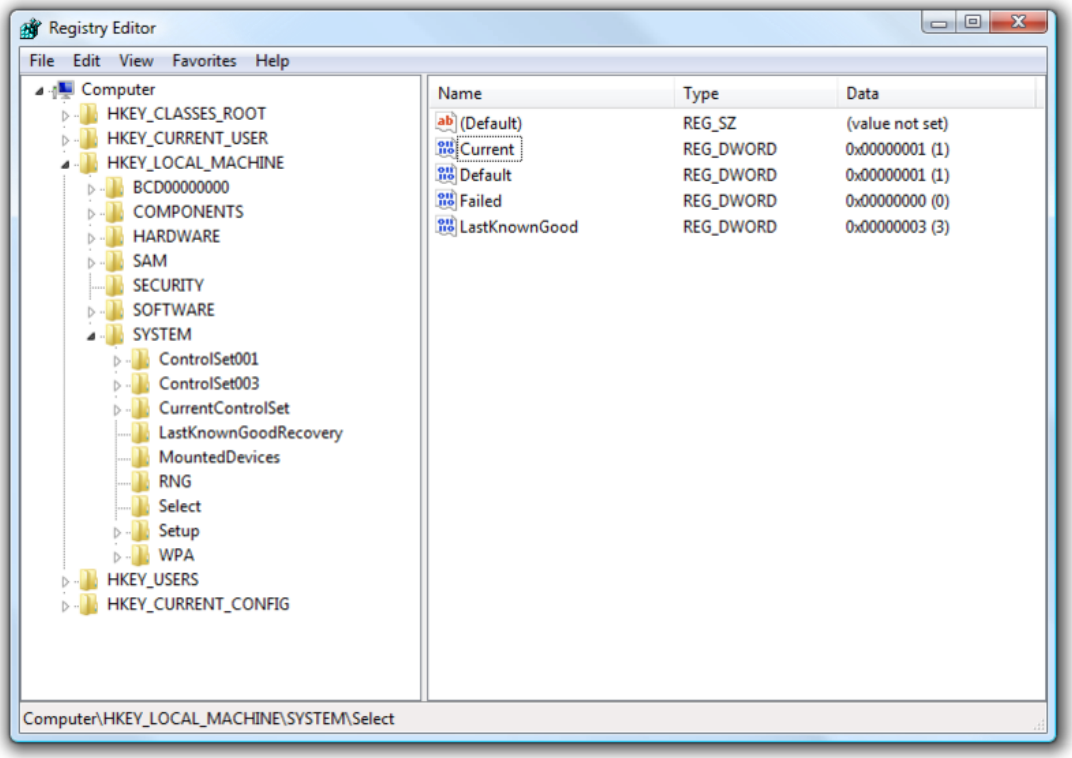
Samotné fungovanie nástroja regedit, ako budeme nástroj Regedit.exe pre zjednodušenie nazývať, je postavené na API funkciách[2]. Množina API funkcií je značne obsiahla a zahŕňa okrem iných aj sadu funkcií, pre prácu s registrami, ktorých zoznam je uvedený v prílohe 1 [21].

Ako vyplýva z prílohy 1., API funkcie orientované na prácu s registrami, ponúkajú rozhranie na pomerne vysokej abstraktnej úrovni prístupu k registrom. Pomocou jednotlivých funkcií totiž možno napríklad priamo pristupovať ku kľúčom registra, nastavovať ich hodnotu či vytvoriť nový kľúč. Je však zrejmé, že vzhľadom na zložitú štruktúru uloženia registrov, žiadna z týchto operácií nie je triviálna a vlastné implementácie jednotlivých funkcií sú zložité. Táto abstrakcia však umožňuje pomerne jednoduché využívanie týchto API funkcií aj v iných aplikáciách, preto napríklad nie je problémom ani vzdialená modifikácia či oprava registrov systému Windows, prostredníctvom aplikácií založených na API funkciách registrov. Tieto funkcie, rovnako ako aj ostatné API funkcie, sú uložené v knižničných súboroch DLL systému Windows a teda sú dostupné všetkým aplikáciám, ktoré ich vo veľkej miere aj využívajú pre ukladanie svojich konfiguračných dát do registrov.

Okrem nástroja regedit sa v operačných systémoch Windows, konkrétne od verzie Windows XP, objavujú aj nástroje reg.exe a regini.exe. Pomocou nástroja reg.exe možno registre modifikovať a čítať podobne ako v nástroji regedit a pomocou nástroja regini.exe možno navyše modifikovať a nastavovať jednotlivé prístupové práva a zabezpečenia registrov, či ich konkrétnych kľúčov a ďalších častí. Oba tieto nástroje využívajú vyššie spomenuté API funkcie.

Nevýhodou nástrojov ako reg.exe alebo regini.exe v porovnaní s nástrojom regedit je ich rozhranie. Kým tieto dva nástroje pracujú len v príkazovom riadku, regedit ponúka prívetivé užívateľské rozhranie, ktoré je zobrazené na obrázku 3.2. Zatiaľ čo na ľavej strane okna nástroja je zobrazená stromová štruktúra načítaných súborov registrov s usporiadaním do kľúčov a podkľúčov, na pravej strane možno sledovať hodnoty jednotlivých kľúčov a podkľúčov aj s ich dátovými typmi a dátami, ktoré obsahujú. Pomocou menu alebo aj jednoduchým kliknutím pravým tlačidlom myši, možno vykonávať jednotlivé operácie či už s kľúčmi, podkľúčmi alebo jednotlivými hodnotami.

[2] API funkcie vytvárajú rozhranie pre tvorbu aplikácií, pričom princíp tohto rozhrania je postavený na správach, prostredníctvom ktorých dochádza ku komunikácii s operačným systémom alebo iným programom, databázou prípadne s komunikačným protokolom. Pod komunikáciou si možno predstaviť napríklad odosielanie ľubovoľných udalostí oknu aplikácie (kliknutie myši, stlačenie klávesy) až po reakciu na túto udalosť.



Obrázok 3.2: Snímok nástroja Regedit.

Okrem výhody grafického užívateľského rozhrania zobrazeného na obrázku 3.2, umožňuje nástroj regedit aj export dát registrov do súboru typu *.reg* a samozrejme aj ich import naspäť. Tento typ súborov, je jednoducho čitateľný a modifikovateľný aj pomocou textových editorov a môže slúžiť ako jednoduchý spôsob prenosu registrov alebo ich častí medzi rôznymi systémami.

Z informácií obsiahnutých v tejto podkapitole vyplýva, že nástroj regedit operačného systému Windows, je komplexným nástrojom pre prácu s registrami. Zahŕňa vytváranie, manipulovanie, premenovávanie a vymazávanie kľúčov, podkľúčov, hodnôt a ich dát, import a export dát registrov, nastavovanie oprávnení či vyhľadávanie názvov kľúčov, hodnôt a dát. Naviazanie na API funkcie a ich vlastná existencia umožňuje ich využitie aj v ďalších aplikáciách, pre ktoré sa tým pádom prístup k registrom stáva taktiež relatívne triviálnou úlohou.

3.8.2 _winreg

Knižnica alebo modul s názvom _winreg je vytvorená v programovacom jazyku Python a umožňuje prístup a prácu s registrami systému Windows. Princíp tohto modulu spočíva vo využívaní API funkcií pre prácu s registrami, ktoré boli spomenuté už v kapitole 3.8.1. Pre používanie stačí tento modul importovať do aplikácie a okamžite je možné pracovať s registrami.

Aktuálna verzia 2.7 ponúka približne polovicu zo všetkých API funkcií pracujúcich s registrami, čo však pre základnú prácu s registrami plne postačuje. Na princípe fungovania tohto modulu je postavený aj ďalší modul s názvom regobj. Tento modul je v porovnaní s modulom _winreg postavený na objektovom prístupe, ktorý prístup k registrom zjednodušuje.

Či už však ide o modul regobj alebo o modul _winreg, obidva fungujú iba v operačnom systéme Windows. Je to logicky spôsobené ich závislosťou od API funkcií, ktoré v iných operačných systémoch nefungujú. Ide teda o moduly závislé od platformy.

3.8.3 Offline Windows Password & Registry Editor

Offline Windows Password & Registry Editor je názvom voľne dostupného nástroja, ktorého autorom je Petter Nordahl-Hagen. Cieľom vývoja tejto aplikácie bolo vytvoriť nástroj pre modifikáciu hesiel operačného systému Microsoft Windows bez potreby byť k tomuto systému prihlásený.

Nakoľko heslá operačného systému Microsoft Windows sú uložené v registroch, bol tento nástroj postupne rozšírený až do dnešnej podoby, keď umožňuje modifikáciu a úpravu registrov.

Jeho nespornou výhodou je možnosť prístupu k registrom operačného systému Windows nezávisle na platforme. Naopak nevýhodou je skutočnosť, že napriek vývoju tohto nástroja už od roku 1997, ide o nástroj, ktorý nie je stále dokončený. Jeho vývoj, úpravy a opravovanie chýb prebieha stále a posledné zmeny v ňom boli vykonané v máji roku 2011 [22].

3.8.4 Wine

Wine je voľne šíriteľný software, ktorý umožňuje na operačných systémoch Linux, Mac, FreeBSD a Solaris spúšťať aplikácie, ktoré by za normálnych okolností fungovali iba v operačnom systéme Windows, a to dokonca bez vytvárania kópie systému Windows v týchto systémoch. Toto je možné vďaka napodobňovaniu funkčnosti jadra systému Windows démonom s názvom wineserver.

Nakoľko Wine vytvára vo vyššie spomenutých systémoch v podstate prostredie systému Windows, je tým pádom v týchto systémoch možné spúšťať aj niektoré z aplikácií samotného Windowsu. Medzi tieto aplikácie patrí aj Regedit popisovaný v kapitole 3.8.1. Vyplýva z toho teda, že v každom operačnom systéme využívajúcom software Wine, je možné pristupovať a pracovať s registrami operačného systému Windows. Skutočnosť je však taká, že ide o registre prostredia, ktoré je softwarom Wine vytvorené, a teda nejde o kompletné registre určitého operačného systému Windows, ktorý je napríklad nainštalovaný na tom istom počítači ako systém, na ktorom beží software Wine.

Rozdiel medzi aplikáciou regedit vo Windowse a aplikáciou regedit v programe Wine je aj ten, že kým vo Windowse sa pracuje s registrami uloženými v binárnych súboroch, v prostredí Wine sa pracuje so súbormi s príponou *.reg*, ktoré sú textové. Hoci je tieto súbory možné editovať aj pomocou textových editorov, odporúča sa používať aplikáciu regedit prostredia Wine hlavne z dôvodu

špeciálneho kódovania pri ukladaní, ktoré by pri editácii v textových editoroch spôsobovalo problémy.

Samotné registre prostredia Wine sú automaticky vytvorené až po prvom spustení tohto softwaru. Následne sú modifikované podľa potreby aplikáciami vo Wine či samotným Wine. Práca s nimi a ich správa pomocou nástroja regedit je totožná s aplikáciou Regedit v skutočnom operačnom systéme Windows, avšak tento nástroj nie je schopný pracovať s binárnymi súbormi, v akých sú skutočné registre uložené. Tento fakt zásadne ovplyvnil aj spôsob implementácie nástroja regedit v prostredí Wine, ktorý v podstate pracuje s textovými súbormi. Náročnosť spracovania je tým pádom omnoho nižšia ako by tomu bolo pri práci s binárnymi súbormi.

Z predchádzajúcich informácií teda jednoznačne vychádza, že software Wine a jeho nástroj regedit nie je schopný pracovať s reálnymi registrami reálneho operačného systému Microsoft Windows, nakoľko nie je schopný pracovať s binárnymi súbormi, v ktorých sú uložené [23].

4 Spôsob uloženia registrov, jeho analýza a návrh modelu pre uloženie dát registrov

Ako vyplýva z kapitoly 3.7, registre systémov Microsoft Windows sú spájané do podregistrov a fyzicky uložené v súboroch podregistrov. Nástroj regedit, popisovaný v kapitole 3.8.1, dáta z týchto súborov spája do prehľadnej hierarchie registrov a umožňuje tým užívateľovi prácu s registrami. Spájanie dát zo súborov podregistrov v nástroji regedit vykonávajú API funkcie, ktoré k týmto systémovým binárnym súborom pristupujú. Keďže však vytváraná aplikácia má byť nezávislá na platforme, použitie API funkcií nepripadá do úvahy, keďže ich funkčnosť v iných operačných systémoch by mohla byť problematická. Taktiež platí, že vytváranie registrov, spôsob ich uloženia ako aj ich správa sú súčasťou operačného systému Microsoft Windows, ktorý je proprietárny, a preto sú aj tieto informácie proprietárne a teda nie sú verejne dostupné.

Keďže oficiálne použitý spôsob uloženia registrov a práca s nimi nie sú voľne prístupné, samotné registre sú uložené v binárnych súboroch a vytváraná aplikácia má byť nezávislá na platforme, získavanie dát z registrov bude potrebné prispôsobiť týmto faktom. Najvhodnejšou metódou sa javí rozbor binárnych súborov podregistrov a ich analýza. Konkrétne pôjde o analýzu uloženia dát v binárnom súbore a zisťovanie ich významu. Analýza by mala viesť k zisteniu, akým spôsobom dáta algoritmicky extrahovať z binárneho súboru a ako s nimi pracovať.

Preto vytvoreniu samotnej aplikácie bude predchádzať zistenie spôsobu uloženia registrov pomocou analýzy súborov podregistrov.

4.1 Analýza

Analýza súborov podregistrov nie je triviálna, nakoľko ide o binárne súbory. Otvorením niektorého z týchto súborov v hexa-editore, akým je napríklad WinHex, získame možnosť náhľadu na súbor vyjadrený v hexadecimálnych hodnotách a tiež v ASCII znakoch. Prehliadaním súboru podregistrov týmto spôsobom, je možné zistiť určité základné informácie o spôsobe rozloženia dát v tomto súbore, avšak presný spôsob uloženia len veľmi ťažko. Omnoho užitočnejšie sú v tomto prípade informácie získané pomocou debuggeru pre systém Windows, s názvom WinDbg. Tento nástroj je voľne prístupný na stránkach Microsoft Windows a s jeho pomocou v kombinácii s príkazmi ako *!reg* alebo *dt nt!* a iné, možno získať značné množstvo užitočných informácií. Ako príklad možno uviesť príkaz *dt nt!_CM_KEY_VALUE*, ktorého výstup zobrazuje tabuľka 4.1. Popisom jednotlivých častí výstupu sa bude zaoberať ďalšia z podkapitol, preto sa mu v tejto časti podrobne venovať nebudeme, avšak

ako je už z príkazu zrejmé, ide o zobrazenie štruktúry resp. spôsobu uloženia *key_value* v pamäti. Zistenie tohto spôsobu uloženia bude veľmi dôležité pri definovaní vlastných pamäťových prvkov, ktoré budú slúžiť pre uloženie jednotlivých prvkov registrov.

Obdobných príkazov ako *dt nt!_CM_KEY_VALUE* je však viac a možno vďaka nim získať veľké množstvo ďalších informácií o spôsobe uloženia jednotlivých prvkov registrov v pamäti. Ako príklady môžeme spomenúť príkazy *dt nt!_CM_KEY_NODE, dt nt!_CM_KEY_SECURITY, dt nt!_CHILD_LIST.*

Tabuľka 4.1: Výstup príkazu dt nt!_CM_KEY_VALUE

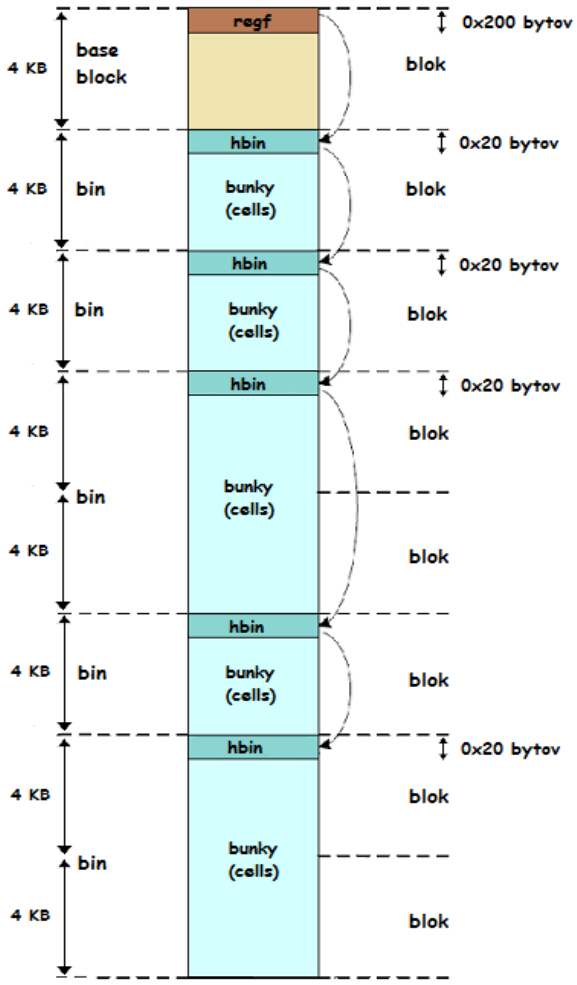
+0x000 Signature	: Uint2B
+0x002 NameLength	: Uint2B
+0x004 DataLength	: Uint4B
+0x008 Data	: Uint4B
+0x00c Type	: Uint4B
+0x010 Flags	: Uint2B
+0x012 Spare	: Uint2B
+0x014 Name	: [1] Wchar

V tomto momente je však potrebné si uvedomiť, že pri používaní nástroja WinDbg a spomenutých príkazov, dochádza k analýze registrov, ktoré sú uložené vo virtuálnej pamäti systému a nie v súboroch podregistrov, akoby sa to dalo očakávať. V skutočnosti totiž nástroj *Configuration Manager* operačného systému Microsoft Windows, ktorý je zodpovedný za implementáciu a správu registrov, ukladá obsah súboru podregistrov do virtuálnej pamäte, kde s týmto obsahom pracuje. V novších verziách *Configuration Manager* mapuje do pamäti iba časti súborov podregistrov, ktoré potrebuje a šetrí tým pamäť pre iné procesy. V každom prípade je dôležité si uvedomiť, že obsah registrov v pamäti nemusí byť totožný s obsahom registrov fyzicky uložených na disku. Analýza obsahu registrov v pamäti však nie je cieľom tejto práce a slúži iba ako pomôcka pri analýze registrov.

Spojením informácií získaných týmto spôsobom analýzy a výsledkov už existujúcich výskumov možno získať pomerne jasnú predstavu o spôsobe uloženia dát v súboroch podregistrov. Popisom zistených informácií sa zaoberá nasledujúca podkapitola.

4.2 Rozloženie súboru podregistrov

V každom zo súborov podregistrov možno sledovať rovnaký vzor resp. spôsob uloženia dát, ktorý je znázornený pre jednoduchšie pochopenie na obrázku 4.1.



Obrázok 4.1: Rozloženie dát v súbore podregistra

Obrázok 4.1 znázorňuje rozdelenie súboru podregistrov, v anglickej terminológii nazývaný *hive*, do blokov. Každý z týchto blokov má veľkosť 4KB (4096 bajtov) a ak je potrebné súbor podregistrov zväčšiť, priestor sa alokuje po blokoch tejto veľkosti.

Prvým blokom v súbore podregistrov, je tzv. *base block* alebo aj základný blok. Obsahuje globálne informácie o súbore podregistrov vrátane signatúry *regf*, ktorá vymedzuje súbor ako súbor podregistrov. Súčasťou *base bloku* je aj hlavička o veľkosti 512 bajtov, do ktorej patrí okrem iného aj spomenutá signatúra. Popis dát patriacich do *base bloku* je uvedený v tabuľke 4.2.

Ako je zrejmé z tabuľky 4.2, množstvo položiek má neznáme využitie, čo by sa mohlo javiť ako problém. V skutočnosti to ale problém nebude, keďže žiadnu z týchto položiek nebude potrebné pre správnu funkčnosť aplikácie využívať.

Najdôležitejšou položkou *base bloku* bude offset prvého *nk* záznamu. Jej význam a využitie bude popísané neskôr.

Tabuľka 4.2: Dáta base bloku

Offset	Veľkosť	Obsah
0x0000	4	Typ bunky: „regf“ = 0x66676572
0x0004	4	Príznak označujúci správnu synchronizáciu
0x0008	4	Príznak označujúci správnu synchronizáciu
0x000c	8	Čas modifikácie v 64 bitovom formáte (Windows filetime)
0x0014	4	Označenie verzie
0x0018	4	Označenie verzie
0x001c	4	Typ
0x0020	4	Formát
0x0024	4	Offset prvého „nk“ záznamu
0x0028	4	Offset na začiatok posledného „hbin“ v súbore
0x002c	4	Cluster [Neznáme využitie]
0x0030	64	Názov hive súboru; maximálna dĺžka 64B
0x0070	396	Rezervované [Neznáme využitie]
0x01fc	4	Kontrolný súčet všetkých dát po túto adresu
0x0200	3528	Rezervované [Neznáme využitie]
0x0fc8	16	ThawTmID [Neznáme využitie]
0x0fd8	16	ThawRmID [Neznáme využitie]
0x0fe8	16	ThawLogID [Neznáme využitie]
0x0ff8	4	BootType [Neznáme využitie]
0x0ffc	4	BootRecover [Neznáme využitie]

Zvyšok súboru podregistrov, nachádzajúci sa za *base blokom*, je rozdelený do blokov, ktoré sú združené do pamäťových celkov s názvom *bin*, čo v preklade znamená kôš. Veľkosť takéhoto koša je minimálne rovná veľkosti bloku. Avšak kôš môže byť tvorený aj viacerými zlúčenými blokmi. V každom prípade teda platí, že veľkosť koša sa musí rovnať násobku veľkosti bloku.

Každý kôš obsahuje hlavičku, ktorej veľkosť je 32 bajtov a nazýva sa *hbin*. Táto hlavička obsahuje základné informácie o dátach v danom koši a jej presnejšie zloženie zachytáva tabuľka 4.3. Na rozdiel od hlavičky *base bloku*, ktorá obsahuje informácie o súbore podregistrov, hlavička koša obsahuje informácie o určitom bloku dát resp. koši obsahujúcom určitý počet blokov.

Častou chybou pri analýze súborov podregistrov a konkrétne pri analýze košov, *bin*, býva skutočnosť, že položka určujúca veľkosť aktuálneho koša, je považovaná za offset nasledujúceho koša. Skúmanie však tento predpoklad nepotvrdzuje, pretože aj posledný kôš v súbore podregistrov obsahuje na tejto položke určitú hodnotu, čo by podľa predpokladov nemal. Keďže už za ním žiadny ďalší kôš neexistuje, mala by táto položka logicky obsahovať hodnotu 0 alebo inú hodnotu, indikujúcu, že za ním už žiadny kôš nie je. Naopak, skúmaním sa mi podarilo zistiť, že táto položka skutočne vyjadruje veľkosť aktuálneho koša, čo opakované skúmanie potvrdilo. Skúmanie hlavičiek košov taktiež ukázalo, že hoci sa položka pre čas modifikácie nachádza v každej hlavičke, v skutočnosti je využitá iba pri hlavičke prvého koša v súbore podregistrov.

Tabuľka 4.3: Dáta koša, bin

Offset	Veľkosť	Obsah
0x0000	4	Typ bunky: „hbin“ = 0x6E696268
0x0004	4	Vzdialenosť medzi prvým „hbin“ a aktuálnym „hbin“
0x0008	4	Veľkosť aktuálneho „hbin“; vždy násobok 4096
0x000c	8	[Neznáme dáta]
0c0014	8	Čas modifikácie v 64 bitovom formáte (Windows filetime)
0x001c	4	[Neznáme dáta]
0x0020	premenlivá	Zoznam buniek použitých pre uloženie dát

Dáta v košoch sa spravidla delia do takzvaných buniek, *cells*. Názov bunka je však len všeobecným pomenovaním pre rôzne druhy buniek, ktoré obsahujú už samotné dáta alebo informácie potrebné pre vyjadrenie dát kľúča alebo hodnoty. Pre bunky vo vzťahu ku košom platí, že žiadna bunka nemôže patriť do viacerých košov resp. že žiadna bunka nemôže ležať naprieč viacerými košmi. Bunky môžu byť druhu kľúč, hodnota, zoznam podkľúčov, zoznam hodnôt alebo môžu popisovať zabezpečenie určitých dát [24]. Každá z týchto buniek, ako aj jej využitie bude presnejšie popísaná v ďalšej podkapitole.

4.3 Bunky v súbore podregistrov

Pomocou informácií obsiahnutých v bunkách, možno pracovať s ľubovoľným prvkom registrov a možno tieto registre zobraziť a vyjadriť hierarchicky, ako sú zobrazené napríklad v nástroji regedit. Bunkami možno popísať okrem samotných kľúčov či hodnôt, aj rôzne vzťahy medzi nimi. Možno popísať, ktoré podkľúče patria ku ktorému kľúču, ktorý kľúč má koľko hodnôt, aké hodnoty to sú, aké dáta obsahujú a podobne.

Pre bunky všeobecne platí, že ich veľkosť je násobkom čísla 8, vďaka čomu dochádza k zarovnaniu pamäte a zjednodušeniu jej správy. Nakoľko každá z buniek má iný význam, je

potrebné ich rozlíšiť už pri ich spracovávaní. Preto prvé dva bajty bunky označujú o aký typ bunky ide. V skutočnosti sa pred týmito bajtmi nachádzajú ešte ďalšie štyri bajty. V nasledujúcom popise jednotlivých buniek však uvádzané nebudú, nakoľko majú pri každej bunke ten istý význam. Tieto bajty totiž vyjadrujú veľkosť bunky, nachádzajúcej sa za nimi, v negatívnom tvare.

4.3.1 Bunka nk

Základnou bunkou vyskytujúcou sa v košoch súborov podregistrov je bunka *nk*, čo je skratka vytvorená z dvojice slov *key node*, čo v preklade znamená uzlový kľúč. Z poradia slov *key node* sa možno domnievať, že z nich vytvorená skratka by mala byť skôr *kn*. Táto domnienka je správna a skratka *nk*, prípadne ďalšie, ktoré budú uvedené neskôr, sa používa z dôvodu uloženia skratky, v opačnom poradí. Takýto spôsob uloženia bajtov obsahujúcich dané znaky alebo číselné hodnoty sa využíva v systémoch x86.

Okrem toho, že bunka *nk* je základnou bunkou, možno povedať, že ide aj o najdôležitejšiu bunku. Bunka *nk* totiž zastupuje v súbore podregistrov samotný kľúč a tiež umožňuje prepojenie jednotlivých kľúčov a hodnôt i vytváranie vertikálnej aj horizontálnej hierarchie medzi kľúčmi.

Ako bolo spomenuté už v kapitole 4.2, *base blok* obsahuje offset koreňového kľúča. Tento fakt bude ďalej využívaný v aplikácii, pretože umožňuje dostať sa z *base bloku* do bunky určitého koša. Touto bunkou bude logicky bunka typu *nk*. Takisto ako hlavičky košov, aj bunky *nk* obsahujú položku, v ktorej je zaznamenaný čas poslednej modifikácie. Bunky *nk* sú jediným druhom bunky, ktorý túto položku obsahuje a jej obsah sa mení pri každej zmene kľúča alebo hodnoty súvisiacej s kľúčom, ktorý je touto bunkou vyjadrený.

Bunka *nk* okrem položky času poslednej modifikácie obsahuje množstvo ďalších veľmi užitočných položiek. Pomocou položky obsahujúcej offset rodičovského kľúča, odkazuje kľúč na svoj rodičovský kľúč, čo je veľmi užitočné pri práci s podkľúčmi. Ďalšie položky odkazujú napríklad na zoznamy hodnôt kľúča alebo na jeho podkľúče. Dôležitou položkou je taktiež položka obsahujúca offset bezpečnostného záznamu kľúča. Pomocou tohto ofsetu sa možno dostať k bezpečnostným záznamom popisujúcim a určujúcim prístupové práva ku kľúču.

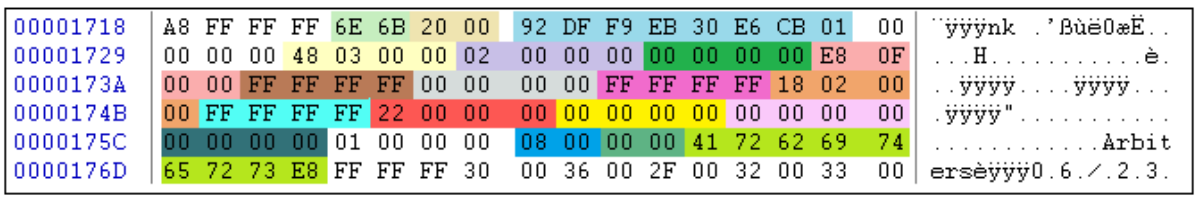
Zaujímavou položkou nachádzajúcou sa v bunke *nk* je položka obsahujúca názov kľúča. Nie je ani tak zaujímavá čo sa týka jej obsahu, keďže obsahuje názov kľúča, zaujímavá je z pohľadu optimalizácie. Súbory registrov sú totiž založené na kódovaní Unicode. Avšak v prípade názvov dochádza pri ich ukladaní k optimalizácii. Optimalizácia je založená na jednoduchom princípe, keď v prípade, že názov obsahuje iba ASCII znaky, je aj uložený vo forme ASCII znakov, čo značne redukuje veľkosť súboru podregistrov. Táto optimalizácia sa využíva aj pri ukladaní názvov hodnôt, ktoré budú popísane v ďalšej časti tejto podkapitoly.

Ďalšie položky bunky *nk* sú popísané v tabuľke 4.4.

Tabuľka 4.4: Bunka nk

Offset	Veľkosť	Obsah	Farba v príklade
0x0000	2	Typ bunky: „nk“ = 0x6B6E	
0x0002	2	Typ kľúča: pre koreňový kľúč obsahuje 0x2C, inak 0x20	
0x0004	8	Čas modifikácie v 64 bitovom formáte (Windows filetime)	
0x000c	4	[Neznáme dáta]	
0x0010	4	Offset rodičovského kľúča („nk“ záznamu)	
0x0014	4	Počet podkľúčov	
0x0018	4	Počet podkľúčov – volatile	
0x001c	4	Offset zoznamu podkľúčov (ak nemá podkľúče, tak 0xffffffff)	
0x0020	4	Offset zoznamu podkľúčov – volatile	
0x0024	4	Počet hodnôt	
0x0028	4	Offset zoznamu hodnôt (ak nemá hodnoty, tak 0xffffffff)	
0x002c	4	Offset bezpečnostného záznamu „sk“	
0x0030	4	Offset class-name	
0x0034	4	Maximálny počet bytov v názve podkľúča	
0x0038	4	Maximálna dĺžka class-name podkľúča	
0x003c	4	Maximálny počet bytov v názve hodnoty	
0x0040	4	Maximálna veľkosť dát hodnoty	
0x0044	4	[Neznáme dáta]	
0x0048	2	Dĺžka názvu kľúča	
0x004a	2	Dĺžka class-name	
0x004c	premenlivá	Názov kľúča	

Pomocou farieb pri jednotlivých položkách v tabuľke 4.4, možno na obrázku 4.2 sledovať hodnoty jednotlivých položiek bunky *nk* konkrétneho kľúča tak, ako sú uložené v binárnom súbore podregistrov a zobrazené už spomínaným nástrojom WinHex. Na obrázku možno tiež sledovať prvé štyri bajty vyjadrujúce veľkosť bunky *nk* v negatívnom tvare, pričom v tejto hodnote sú započítané aj samotné štyri bajty vyhradené pre údaj o veľkosti. Z negatívneho tvaru veľkosti bunky možno jej skutočnú veľkosť dostať rozdielom hodnoty 0x100000000, čo je maximálna hodnota, ktorú možno vyjadriť na 32 bitoch, a hodnoty veľkosti v negatívnom tvare. Po vykonaní tohto rozdielu dostaneme skutočnú veľkosť bunky, ktorá bude deliteľná číslom 8, z už spomínaných dôvodov.



Obrázok 4.2: Bunka *nk* v súbore podregistrov

4.3.2 Bunka sk

Ďalšou bunkou nachádzajúcou sa v priestore koša je bunka *sk*. Jej skratka pochádza zo slov *key security*, čo v preklade znamená zabezpečenie kľúča. Je teda zrejmé, že ide o druh bunky, ktorý bol spomenutý už vyššie. Jeho význam spočíva v určovaní a vymedzovaní prístupových práv, na základe vlastníctva, k jednotlivým kľúčom, hodnotám a podkľúčom. Každá *nk* bunka resp. kľúč, obsahuje offset bezpečnostného kľúča. Nie je však pravidlom, že každý kľúč obsahuje offset na iný bezpečnostný kľúč. Práve naopak, bezpečnostných kľúčov je o poznanie menej než bežných kľúčov, čo je však spôsobené tým, že určité skupiny kľúčov majú nastavené rovnaké prístupové práva a teda by bolo zbytočné, aby existovali opakujúce sa rovnaké bezpečnostné kľúče. Znamená to teda, že bezpečnostný kľúč môže byť využívaný väčší počtom kľúčov. A hoci samotný bezpečnostný kľúč neobsahuje informáciu o tom, ktorými kľúčmi je využívaný, obsahuje informáciu o počte kľúčov, ktoré na neho odkazujú resp. ho využívajú. Popis ďalších položiek bunky *sk* obsahuje tabuľka 4.5.

Tabuľka 4.5: Bunka sk

Offset	Veľkosť	Obsah
0x0000	2	Typ bunky: „sk“ = 0x6B73
0x0002	2	[Neznáme dáta]
0x0004	4	Offset predchádzajúceho „sk“ záznamu
0x0008	4	Offset nasledujúceho „sk“ záznamu
0x000c	4	Počítadlo použití „sk“ záznamu
0x0010	4	Veľkosť „sk“ záznamu
0x0014	premenlivá	Dáta, obsahujúce bezpečnostné nastavenia; ich veľkosť je daná predchádzajúcou položkou

4.3.3 Bunka vk

Bunkou slúžiacou pre vyjadrenie hodnoty kľúča, je bunka typu *vk*, ktorej názov je skratkou slov *key value* a znamená hodnota kľúča. Spojením obsahu tejto bunky s obsahom bunky *nk*, ktorá na túto bunku *vk* ukazuje, dostaneme dvojicu kľúč – hodnota, tak, ako ju možno vidieť aj v nástroji regedit. Ako však dochádza k prepojeniu kľúča vyjadreného bunkou *nk* s určitou hodnotou vyjadrenou bunkou *vk*? Bunka *nk* obsahuje položku s offsetom zoznamu hodnôt, ktorá logicky odkazuje na miesto v súbore podregistrov, kde je zoznam hodnôt daného kľúča uložený. Zoznam hodnôt je jediným pamäťovým prvkom v súbore podregistrov, ktorý je využívaný na odkazovanie sa niekam a pritom nemá žiadnu signatúru. V podstate ide len o miesto v pamäti, ktoré je uvedené štyrmi bajtmi vyjadrujúcimi veľkosť zoznamu hodnôt v negatívnom tvare a samotnými záznamami hodnôt. Tieto záznamy hodnôt však v skutočnosti neobsahujú informácie o hodnotách, pre každú z hodnôt kľúča je

tento záznam tvorený len štyrmi bajtmi, ktoré vyjadrujú offset, na ktorom možno nájsť konkrétnu bunku *vk* popisujúcu konkrétnu hodnotu. Ide teda len o pomocný prvok, ktorý sprostredkováva prepojenie kľúča a jeho hodnôt.

Samotná bunka *vk* obsahuje podobne ako aj predchádzajúce bunky viacero položiek. Význam niektorých, ako dĺžka názvu hodnoty či názov hodnoty, je očakávaný, iné je potrebné podrobnejšie vysvetliť.

Medzi položky s nie celkom zrejmým významom patrí položka určujúca typ dát. Jej význam však úzko súvisí s položkou udávajúcou offset dát hodnoty, preto bude táto vysvetlená ako prvá. Položka offsetu dát hodnoty má podobný význam ako položka offsetu zoznamu hodnôt v bunke *nk*. Táto položka totiž ukazuje na miesto v pamäti, kde sú uložené dáta patriace k danej hodnote. Na tomto mieste sa potom budú nachádzať samotné dáta hodnoty uvedené štyrmi bajtmi udávajúcimi veľkosť týchto dát v negatívnom tvare, pričom táto veľkosť, rovnako ako v predchádzajúcom prípade, zahŕňa aj štyri bajty slúžiace pre uloženie údaju o veľkosti. Takýto mechanizmus uloženia dát hodnoty je zavedený preto, aby bunky *vk* neboli príliš rozsiahle, čo by značne skomplikovalo ich správu a teda aj správu pamäti súboru podregistrov. V istých prípadoch je však veľkosť dát hodnoty taká malá, že ju možno uložiť na miesto, kde by v normálnom prípade bol uložený offset odkazujúci na tieto dáta. Ak je veľkosť dát napríklad štyri bajty, možno tieto dáta uložiť na miesto položky udávajúcej offset dát, keďže táto položka má rovnako štyri bajty. A práve na identifikáciu toho, akým spôsobom sú dáta hodnoty uložené, slúži položka obsahujúca informáciu o type dát. Ak táto položka obsahuje hodnotu 0x0000, bude položka obsahujúca offset dát obsahovať skutočne offset dát a teda k dátam hodnoty sa dostaneme len cez tento odkaz. Keď je ale obsah položky rovný číslu 0x8000, dáta sú uložené priamo v priestore položky vyhradenej pre offset dát.

Z ďalších položiek je vhodné spomenúť položku obsahujúcu príznak pomenovania hodnoty. Pri hodnotách je totiž na rozdiel od kľúčov možné, aby neboli pomenované. Ak však nie sú pomenované, je potrebné to určitým spôsobom indikovať, hlavne z dôvodu zjednodušenia práce s hodnotami. Preto, ak hodnota nie je pomenovaná, obsahuje táto položka hodnotu 0x00000000, v opačnom prípade obsahuje hodnotu 0x01000000. Nepomenované hodnoty možno sledovať aj v nástroji regedit, kde sa namiesto ich nezadaného mena používa označenie *Default*.

Poslednou položkou, ktorá by nemusela byť celkom jasná je položka udávajúca typ hodnoty. V tomto prípade však nejde o nič nové a táto položka obsahuje informáciu o tom, akého typu je daná hodnota, keďže ako bolo spomenuté už v kapitole 3.7, každá hodnota musí byť určitého typu. Pričom pod pojmom typ hodnoty je potrebné chápať typy, ktoré boli popísané v kapitole 3.7 a ich zoznam i konkrétnejší popis obsahuje tabuľka 3.2.

Popis zvyšných položiek bunky *vk*, ktorú budeme nazývať aj hodnota, sa nachádza v nasledujúcej tabuľke 4.6.

Tabuľka 4.6: Bunka vk

Offset	Veľkosť	Obsah
0x0000	2	Typ bunky: „vk“ = 0x6B76
0x0002	2	Dĺžka názvu hodnoty
0x0004	2	Dĺžka dát hodnoty
0x0006	2	Typ dát
0x0008	4	Offset dát hodnoty alebo pri niektorých typoch dáta hodnoty
0x000c	4	Typ hodnoty
0x0010	4	Príznak určujúci, či je hodnota pomenovaná alebo nie.
0x0014	premenlivá	Názov hodnoty

4.3.4 Bunky lf, lh, li, ri

Posledným druhom buniek, nachádzajúcim sa v košoch súborov podregistrov, sú bunky umožňujúce vytvárať vertikálnu hierarchiu medzi kľúčmi a to tak, že umožňujú existenciu podkľúčov určitého kľúča. Podobne ako pri hodnotách, existuje aj pre kľúče medzi položkami bunky *nk* konkrétna položka, ktorá obsahuje offset zoznamu podkľúčov. Dá sa teda predpokladať, že adresovanie podkľúčov bude podobné ako bolo adresovanie hodnôt. Pre adresovanie podkľúčov však existujú štyri druhy buniek, z čoho vyplýva, že samotné adresovanie podkľúčov a odkazovanie na ne z nadradených kľúčov bude zložitejšie. Skúmanie súborov podregistrov, ako aj jednotlivých buniek, ich súvislostí a vzťahov ukázalo, že možno vymedziť dva spôsoby adresovania podkľúčov z kľúča. Nejde však o spôsoby v zmysle, že sa používa buď jeden alebo druhý. Oba spôsoby sú využívané v súboroch podregistrov súčasne a majú svoje opodstatnenie.

Jednoduchšou z alternatív adresovania podkľúčov je využitie niektorej z buniek *lf* alebo *lh*. O jednoduchší spôsob adresovania ide preto, lebo z položky bunky *nk*, ktorá obsahuje offset zoznamu hodnôt, je možné dostať sa priamo na jednu z buniek *lf* alebo *lh*, z ktorých sa možno dostať na bunku podkľúča. Obe tieto bunky sú si veľmi podobné a obsahujú okrem signatúry aj informáciu o počte podkľúčov, na ktoré odkazujú. Je zrejmé, že tieto bunky nebudú obsahovať informácie o podkľúčoch, keďže už vieme, že každý kľúč je popísaný bunkou *nk*. Preto teda bunky *lf* aj *lh* obsahujú offsety na bunky podkľúčov. Rozdiel medzi nimi je v tom, že kým v bunke *lf* po offsete podkľúča nasledujú štyri bajty zaplnené prvými štyrmi znakmi názvu podkľúča, v prípade bunky *lh* je to inak.

Bunky *lh* sú novšie ako bunky *lf*, boli zavedené pravdepodobne po verzii operačného systému Windows 2000. Rozdiel medzi nimi spočíva v tom, že v bunkách *lh* sa po offsete podkľúča nenachádzajú prvé štyri znaky z názvu podkľúča, ale na týchto štyroch bajtoch je uložená číselná hodnota. Táto hodnota je vypočítavaná špeciálnym hashovacím algoritmom založeným na názve daného podkľúča. Preto, ak by sa zmenil názov podkľúča, bolo by potrebné túto hodnotu opäť vypočítať a zmeniť, aby bola zachovaná konzistencia informácií.

Zmyslom týchto štyroch bajtov, či už v podobe číselného výsledku hashovacieho algoritmu alebo v podobe prvých štyroch znakov názvu podkľúča, je rýchlosť. Pretože napríklad v prípade potreby vyhľadania určitého podkľúča by v situácii ak by spomínané štyri bajty neexistovali, bolo potrebné za použitia offsetu pristupovať k jednotlivým *nk* bunkám podkľúčov. A až na základe ich obsahu, prostredníctvom porovnania s položkou názvu kľúča, by bolo možné zistiť, či ide o požadovaný podkľúč alebo nie. Tým, že bunky *lf* a *lh*, obsahujú spomínané štyri bajty pre každý z podkľúčov, je možné sa zbytočným prístupom k *nk* bunkám podkľúčov vyhnúť. Zhoda mien sa skontroluje na základe prvých štyroch znakov názvu alebo prostredníctvom porovnania výstupov hashovacích algoritmov. Tým sa šetrí čas a práca s registrami je rýchlejšia.

Nižšie uvedené tabuľky 4.7 a 4.8 zobrazujú celý obsah buniek *lf* a *lh* aj s popismi jednotlivých položiek.

Tabuľka 4.7: Bunka lf

Offset	Veľkosť	Obsah
0x0000	2	Typ bunky: „lf“ = 0x666C
0x0002	2	Počet podkľúčov
0x0004	4	Offset podkľúča, čiže „nk“ záznamu
0x0008	4	Prvé 4 znaky názvu odkazovaného „nk“ záznamu
[nasledujú informácie o ďalších podkľúčoch vyjadrené opakovane prostredníctvom predchádzajúcich 2 položiek; počet opakovaní je daný počtom podkľúčov]		

Tabuľka 4.8: Bunka lh

Offset	Veľkosť	Obsah
0x0000	2	Typ bunky: „lh“ = 0x686C
0x0002	2	Počet podkľúčov
0x0004	4	Offset podkľúča, čiže „nk“ záznamu
0x0008	4	Hash hodnota názvu odkazovaného „nk“ záznamu
[nasledujú informácie o ďalších podkľúčoch vyjadrené opakovane prostredníctvom predchádzajúcich 2 položiek; počet opakovaní je daný počtom podkľúčov]		

Zložitejšou alternatívou adresovania podkľúčov je adresovanie pomocou buniek *ri* a *li*. V tomto prípade nejde o použitie jednej alebo druhej bunky, ale o použitie oboch súčasne. Väčšia zložitosť tejto alternatívy adresovania podkľúčov spočíva v existencii ďalšieho medzičlánku medzi kľúčom a podkľúčom. Týmto medzičlánkom je bunka *ri*.

Adresovanie týmto spôsobom funguje tak, že položka kľúča obsahujúca offset zoznamu podkľúčov obsahuje offset bunky *ri*. Bunka *ri* obsahuje offsety buniek *li* a až tieto bunky obsahujú offsety vlastných podkľúčov.

Táto metóda adresovania sa využíva v prípade, ak má kľúč veľké množstvo podkľúčov čo by spôsobilo enormnú veľkosť bunky *lf* alebo *lh*. Množstvo podkľúčov, považované v tomto prípade za kritické, je s najväčšou pravdepodobnosťou 512. Pokusy totiž preukázali, že maximálny počet podkľúčov, ktorý adresuje bunka *li* je 512, avšak v každom z prípadov existencie bunky *li*, nikdy nešlo o osamotenú *li* ale vždy o viac takýchto buniek, ktoré spolu adresovali viac ako 512 podkľúčov. Preto, keďže počet adresovaných podkľúčov bunkami *li* nikdy nebol menší ako 512, možno predpokladať, že takýto počet sú schopné adresovať aj bunky *lh* a *lf*.

Pri tomto množstve teda namiesto adresovania podkľúčov priamo z jednej z buniek *lf* alebo *lh*, dochádza k vytvoreniu bunky *ri*. Táto bunka obsahuje informáciu o počte buniek *li*, pomocou ktorých sú adresované všetky podkľúče daného kľúča a taktiež obsahuje položky s ofsetmi jednotlivých *li* buniek.

Popis položiek bunky *ri* obsahuje tabuľka 4.9.

Tabuľka 4.9: Bunka ri

Offset	Veľkosť	Obsah
0x0000	2	Typ bunky: „ri“ = 0x6972
0x0002	2	Počet „li“ záznamov
0x0004	4	Offset korešpondujúceho „li“ záznamu
[nasledujú offsety ďalších „li“ záznamov, ktorých počet je daný počtom „li“ záznamov]		

Bunka *li* ako bolo vyššie spomenuté, obsahuje offsety jednotlivých *nk* buniek podkľúčov. Okrem toho obsahuje aj informáciu o ich počte, ako to je znázornené v tabuľke 4.10.

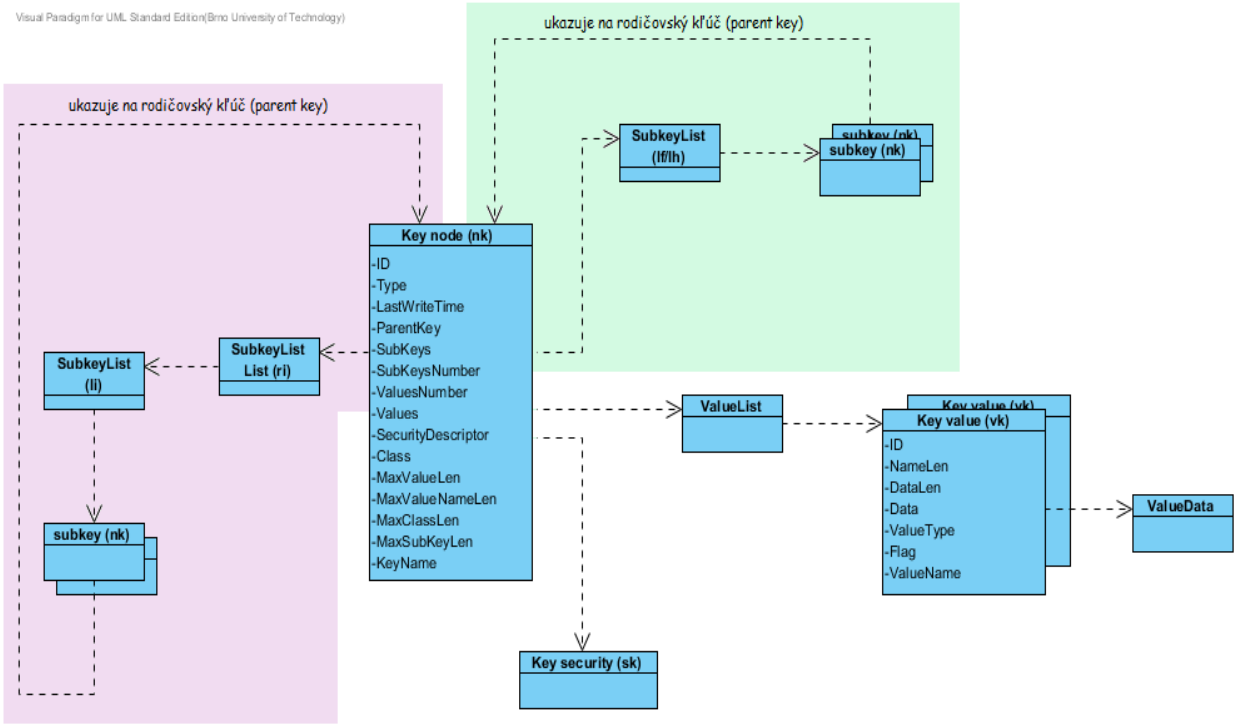
Tabuľka 4.10: Bunka li

Offset	Veľkosť	Obsah
0x0000	2	Typ bunky: „li“ = 0x696C
0x0002	2	Počet podkľúčov
0x0004	4	Offset podkľúča, čiže „nk“ záznamu
[nasledujú offsety ďalších „nk“ záznamov, ktorých počet je daný počtom podkľúčov]		

Vo vzťahu k bunkám umožňujúcim existenciu podkľúčov je dôležité spomenúť aj ďalšiu existujúcu optimalizáciu vyhľadávania podkľúčov. Okrem optimalizácie pomocou buniek *lf* a *lh*, sa používa aj optimalizácia založená na usporiadaní offsetov na základe abecedného poradia názvov podkľúčov, na ktoré tieto offsety odkazujú. Táto optimalizácia spočíva v možnosti využiť binárne vyhľadávanie pre vyhľadanie podkľúča v zozname offsetov podkľúčov. Vďaka tejto možnosti sa pri hľadaní podkľúča najskôr porovná hľadaný podkľúč s podkľúčom, na ktorý odkazuje stredný offset zoznamu offsetov podkľúčov. Ak je názov tohto podkľúča abecedne za názvom podkľúča hľadaného,

ďalšie vyhľadávanie bude pokračovať v prvej polovici zoznamu offsetov obdobným spôsobom. V opačnom prípade sa bude obdobným spôsobom pokračovať vo vyhľadávaní v druhej polovici zoznamu offsetov.

Lepšiu predstavu o vzťahoch a prepojeniach jednotlivých buniek v súboroch podregistrov si možno vytvoriť pomocou obrázka 4.3, ktorý farebne odlišuje dve existujúce alternatívy adresovania podkľúčov. Priamu, pomocou buniek *lf* a *lh*, a nepriamu, pomocou buniek *ri* a *li*.



Obrázok 4.3: Prepojenia buniek v súbore podregistrov

Z popisu uloženia registrov v súboroch podregistrov je zrejmé, že ich uloženie skutočne nie je jednoduché. Adresovanie kľúčov a hodnôt z jedného kľúča môže byť jednoduché, avšak pri zväčšovaní stupňa vertikálnej hierarchie prostredníctvom podkľúčov a ich podkľúčov atď., sa prehľadnosť stráca a použitie, či spracovanie jednotlivých správnych buniek je zložitejšie. Toto bude jeden z hlavných problémov, na ktoré bude treba dbať pri implementácii, ktorú popisuje nasledujúca kapitola.

Ako zdroje k tejto kapitole okrem vlastného výskumu boli použité [25] [26] [27] [28].

4.4 Návrh dátového modelu pre uloženie dát získaných z registrov

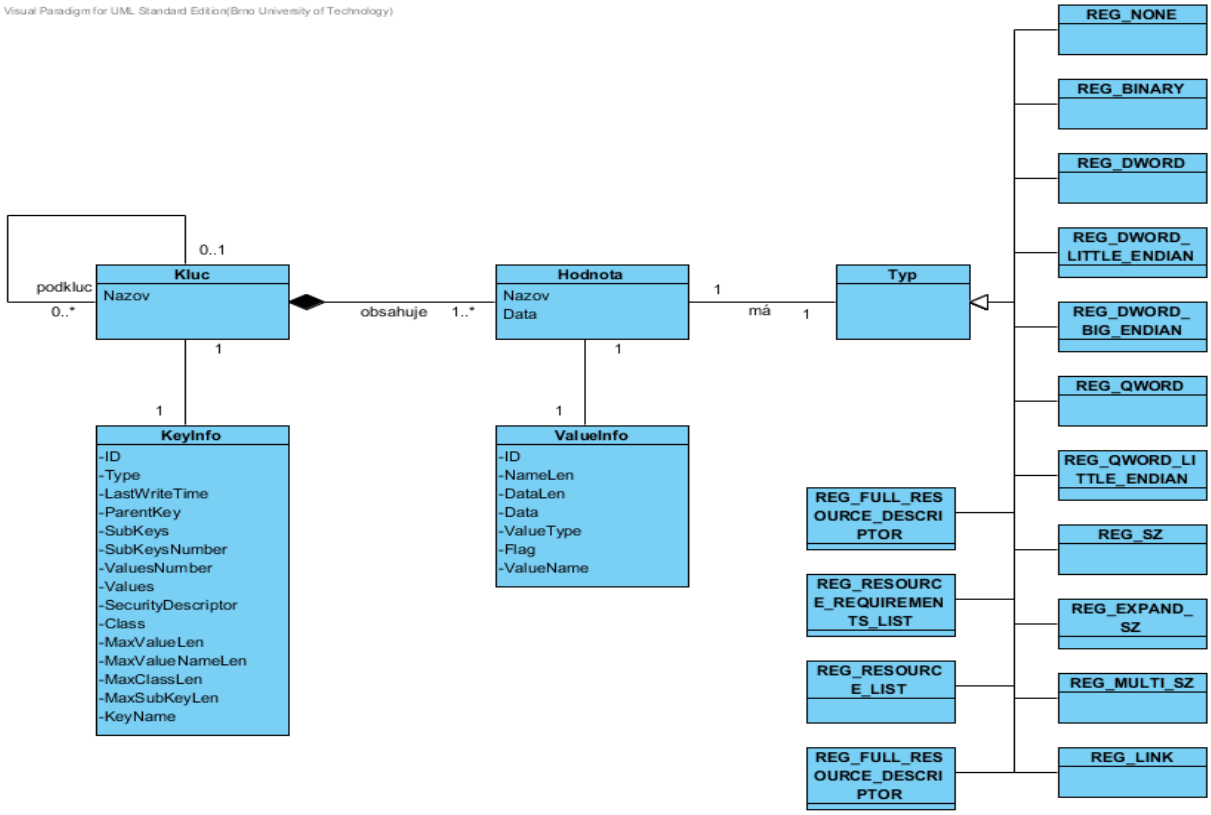
Na základe výsledkov analýzy spôsobu uloženia registrov, ktorá bola vykonaná v predchádzajúcich podkapitolách, možno navrhnúť dátový model pre uloženia dát získaných z registrov. Tento model bude taktiež korešpondovať s pravidlami, ktoré platia pre registre a boli popísané v kapitole 3.7.

Hoci z kapitoly 4.3 jasne vyplýva, že jednotlivé bunky súborov podregistrov spolu úzko súvisia, pri návrhu modelu nebude potrebné pracovať so všetkými druhmi buniek, ktoré boli definované. Hlavným dôvodom je fakt, že model má slúžiť pre uloženie dát registrov, ktoré budú ďalej analyzované v súvislosti so zmenami spôsobenými škodlivým softwarom. V tomto kontexte nie je potrebné vytvárať model, ktorý by zabezpečoval uloženie všetkých druhov buniek, nakoľko niektoré pre túto analýzy nebudú potrebné. Takými budú bunky *lf*, *lh*, *li* a *ri* slúžiace na adresovanie podkľúčov, bunka pre zabezpečenia kľúča *sk*, a bunky *hbin* a *regf*. Pre spracovanie súboru podregistrov sú tieto bunky nenahraditeľné, avšak pre účel analýzy sú najdôležitejšie bunky *nk* a *vk*, ktorých uloženie bude model zabezpečovať.

Návrh modelu pre uloženie dát získaných z registrov vytvorený na základe predchádzajúcej špecifikácie je zobrazený na obrázku 4.4. Z návrhu vyplýva, že každý kľúč registrov, či už pôjde o koreňový kľúč alebo ostatné kľúče štruktúry registrov, bude obsahovať buď žiaden alebo jeden a viac podkľúčov. Tým, že kľúč obsahuje ďalšie kľúče, sa stáva jeho nadradeným kľúčom. Zároveň pre každý z vytvorených kľúčov bude platiť, že obsahuje určité hodnoty. Pre registre platí, že každý kľúč musí obsahovať aspoň jednu hodnotu avšak môže obsahovať neobmedzený počet hodnôt, čo musí rešpektovať aj model pre uloženie dát a jasne to vyjadruje aj násobnosť v diagrame tried.

Trieda *Hodnota* nemohla byť zahrnutá do triedy *Kluc* ako atribút, keďže na rozdiel od atribútu *Nazov* vyjadruje zložitejšie informácie. Atribútmi triedy hodnota sú *Nazov* a *Data*. Nemožno však zabudnúť na pravidlo platné pre hodnoty registrov, že každá hodnota musí mať aj určitý typ. Typ hodnoty je v modeli vyjadrený ako trieda *Typ*, ktorá je vo vzťahu generalizácie s ďalšími triedami. Tieto triedy vyjadrujú jednotlivé hodnoty, ktoré môže *Typ* nadobúdať. Trieda *Hodnota* samozrejme nemôže existovať sama a musí byť súčasťou kľúča.

Ku každému objektu z tried *Kluc* a *Hodnota*, náleží vždy práve jeden objekt tried *KeyInfo* resp. *ValueInfo*. Ide o abstraktné triedy, ktoré budú používané, ale priamo ich nebude vidno, resp. nebudú dôležité z hľadiska uloženia kľúčov a hodnôt. Ide totiž o jednotlivé položky buniek *nk* resp. *vk* tak, ako boli popísané v kapitole 4.3.1 resp. 4.3.3. Z hľadiska uloženia kľúčov a hodnôt sú síce obe bunky dôležité, avšak nie sú dôležité všetky ich položky, preto nemožno atribúty tried *KeyInfo* resp. *ValueInfo* zahrnúť medzi atribúty tried *Kluc* a *Hodnota*.



Obrázok 4.4: Dátový model pre uloženie dát registrov

Z hľadiska popisu chovania škodlivého kódu bude dôležité implementovať funkciu pre export dát z binárneho súboru podregistrov do súboru v zrozumiteľnej forme. Exportom dát súborov podregistrov normálneho systému a súborov podregistrov napadnutého systému získame dva typy súborov, ktorých porovnaním bude možné odhaliť zmeny vykonané škodlivým softwarom v registroch systému. Uloženie obsahu registrov v týchto súboroch bude korešpondovať s vyššie popísaným modelom pre uloženie dát exportovaných z registrov.

Implementáciou tejto funkcie ako aj ďalších dôležitých funkcií sa budú zaoberať ďalšie kapitoly.

5 Implementácia

Implementácia aplikácie nezávislej na platforme pre prístup k registrom je založená na teórii popísanej v kapitole 4. a pre jej realizáciu bol zvolený programovací jazyk C/C++.

5.1 Načítanie a export súboru podregistrov

Samotnému načítaniu súboru podregistrov bude predchádzať načítanie jeho obsahu do buffera. Tým, že celý obsah tohto súboru bude uložený v bufferi, obmedzíme množstvo prístupov k súboru. Bude totiž možné pracovať s bufferom a nebude potrebné zakaždým pristupovať do súboru.

Preto, aby bolo možné súbor načítať, je potrebné sa dostať ku koreňovému kľúču, pomocou ktorého už nebude problém dostať sa aj k ďalším kľúčom a hodnotám. Ako je zrejmé z tabuľky 4.2, ku koreňovému kľúču je možné sa dostať pomocou informácii z *base bloku*. Ako však tieto informácie získať? V aplikácii si vytvoríme štruktúry, ktoré budú korešpondovať s obsahmi buniek uvedenými v kapitolách 4.2 a 4.3. Naplnenie týchto štruktúr bude spočívať v jednoduchom priradení obsahu buffera do tejto štruktúry. Tým, že veľkosti štruktúr budú obmedzené, naplnia sa iba takým množstvom dát z buffera, ktoré sa rovná ich veľkosti. Tento princíp je veľmi jednoduchý a bude využívaný pri práci so všetkými bunkami v celej aplikácii.

Týmto spôsobom si naplníme štruktúru vytvorenú pre *base blok*. Na základe jej položky uloženej na offsete 0x0024, vieme zistiť offset koreňového kľúča celého súboru podregistrov. Pred samotným načítaním koreňového kľúča, je však potrebné si uvedomiť jednu dôležitú vec týkajúcu sa offsetu, ktorá nás bude sprevádzať celou aplikáciou. Offset koreňového kľúča, ako aj offsetov všetkých ostatných buniek v súbore podregistrov, nevyjadruje vzdialenosť od začiatku súboru, ako by sa dalo očakávať. Ide totiž o vyjadrenie vzdialenosti od začiatku prvého koša, čo z pohľadu buniek je vlastne začiatok priestoru, kde začínajú existovať, nakoľko *base blok* bunky neobsahuje. Ako už vieme z kapitoly 4.2, na začiatku súboru podregistrov sa nachádza *base blok*, ktorého veľkosť, keďže je to blok, je 4 KB alebo aj 4096 bajtov. Hneď za ním začína prvý kôš a teda kôš, od ktorého začiatku sú offsety buniek udávané. Z toho potom vyplýva, že ak budeme chcieť načítať nejakú bunku, budeme sa musieť v bufferi posunúť nie len o jej offset ale o offset sčítaný s hodnotou 4096.

Po týchto úvodných krokoch sme schopný sa dostať už k jednotlivým ďalším kľúčom a ich hodnotám. Cyklicky teda načítavame jednotlivé kľúče a hodnoty, ich podkľúče a tak ďalej, čo spôsobuje aj zmeny úrovne zanorenia kľúčov. Obsah súboru podregistrov je samozrejme možné aj uložiť so súboru, čím získame prehľadne zobrazenú celú hierarchiu kľúčov a ich hodnôt. Aby toto zobrazenie bolo skutočne prehľadné a bolo z neho možné získať určitú predstavu o tom, kde sa ktorý kľúč nachádza, je potrebné udávať pri kľúčoch celú cestu k nim. Pod pojmom cesta si treba predstaviť jednoducho okrem názvu daného kľúča aj názvy kľúčov jemu nadradených, čiže cesta

bude v podstate vyjadrovať umiestnenie kľúča v hierarchii kľúčov. Toto však nie je možné dosiahnuť jednoduchým vypisovaním názvu kľúč z bunky *nk* jemu prislúchajúcej. Preto je v aplikácii využívaný vektor, do ktorého sa pri každom zanorení ukladá názov kľúča, v ktorom práve dochádza vyhľadávanie hodnôt a jeho podkľúčov. Pri vynorení sa kľúč, z ktorého sme sa práve vynorili, z tohto vektora maže. Týmto spôsobom je možné vždy evidovať všetky nadradené kľúče práve spracovávaného kľúča.

Možnosť načítania kľúčov a hodnôt súboru podregistrov a ich prehľadného exportu resp. uloženie do súboru, je veľmi dôležité hlavne z dôvodu ďalšieho využitia. Obsahy takýchto súborov totiž budú slúžiť pre porovnávanie pôvodných obsahov súborov podregistrov so súbormi podregistrov systémov, ktoré boli napadnuté škodlivým softwarom.

5.2 Vyhľadávanie kľúčov a hodnôt

Optimalizácie pri vyhľadávaní kľúčov, ktoré boli popísané v kapitole 4.3, aplikácia nebude obsahovať. Vyhľadávanie kľúčov, ako aj hodnôt bude prebiehať v pomocných vektoroch, ktoré sa počas načítavania obsahu súboru podregistrov postupne napĺňajú. Jeden vektor bude obsahovať všetky kľúče a druhý všetky hodnoty súboru podregistrov. Toto eliminuje opätovnú prácu s bufferom, vypočítavanie offsetov jednotlivých buniek a ďalšie činnosti, ktoré by súviseli s vyhľadávaním v bufferi a zbytočne by zaberali čas.

Obe vyhľadávania, či už ide o vyhľadávanie hodnôt ako aj vyhľadávanie kľúčov, majú svoje špecifiká. Pri vyhľadávaní hodnôt nestačí pracovať s názvom hodnoty, ale je potrebné pracovať aj s názvom kľúča, ku ktorému hodnota patrí a s cestou k nemu. Dôvodom je skutočnosť, že v súbore podregistrov sa názvy hodnôt opakujú, pretože musia byť jedinečné iba v rámci kľúčov. V samotnej hierarchii kľúčov sa totiž môžu ľubovoľne opakovať.

Na druhej strane pri vyhľadávaní kľúčov je potrebné brať do úvahy aj dĺžku ich názvu. Jednoduché porovnanie zadaného kľúča aj s cestou k nemu s uloženými cestami kľúčov nebude postačujúce, pretože existujú kľúče, ktorých názvy sú do určitej dĺžky zhodné. Keby sa nebrala do úvahy ich dĺžka, dochádzalo by k nesprávnemu vyhľadaniu kľúča.

5.3 Modifikácia hodnôt

Ďalšou z implementovaných funkcií je modifikácia dát hodnôt. Pomocou tejto funkcie možno modifikovať dáta zadanej hodnoty v zadanom kľúči. Pri implementácii tejto funkcie si bolo potrebné uvedomiť, aké rozdiely medzi jednotlivými hodnotami existujú. Smerodajný je v tomto prípade rozdiel týkajúci sa typov dát, ktoré boli popísané v kapitole 3.7. Typ dát totiž značne ovplyvňuje spôsob modifikácie dát.

Napríklad hodnota typu REG_DWORD je uložená na ôsmich bajtoch a obsahuje určitú číselnú hodnotu. Spomenutých osem bajtov určuje, aké maximálne a aké minimálne číslo tu môže byť uložené. Tým pádom užívateľ môže chcieť túto hodnotu modifikovať na nejakú hodnotu mimo vymedzený rozsah, avšak počet bajtov vymedzených na uloženie tohto typu hodnoty nezmení. Okrem toho, ak sa užívateľ pokúsi hodnotu modifikovať na hodnotu mimo rozsah, táto hodnota bude automaticky zapísaná ako jedna z hraničných hodnôt intervalu možných hodnôt v závislosti od toho, ku ktorej hraničnej hodnote bude zadaná hodnota bližšie.

V prípade hodnôt reťazcového typu, ako napríklad REG_SZ, je však situácia značne odlišná. Pri tomto type hodnôt totiž nemožno napevno stanoviť, koľko bajtov môže byť použitých pre ich uloženie. Preto je potrebné ošetriť situácie, keď užívateľ existujúce dáta modifikuje a ich dĺžka sa oproti pôvodnej zmení. V prípade, že sú novo zadané dáta väčšie ako dáta pôvodné, aplikácia najskôr zistí, či za miestom uloženia pôvodných dát neexistuje voľný priestor, vďaka ktorému by nové dáta bolo možné uložiť na pôvodné miesto s tým, že by zaplnili aj nasledujúce voľné miesto. Ak toto možné nie je, aplikácia musí nájsť nový voľný priestor, kam by sa dáta danej dĺžky zmestili. Zároveň však vymaže pôvodné dáta a upraví offset dát a ich dĺžku v príslušnej *vk* bunke.

Ak užívateľ modifikoval pôvodné dáta, kratším reťazcom, čiže kratšími dátami, uložia sa tieto dáta na miesto dát pôvodných. Pred uložením sa ešte nové dáta doplnia do dĺžky pôvodných dát znakom '\0', čo v podstate bude znamenať, že po uložení, sa pôvodné dáta, ktoré neboli prepísané novými dátami, vymažú. Samozrejme je taktiež potrebné modifikovať offset dát a ich dĺžku.

Pri ukladaní reťazcových dát si treba uvedomiť, že pri ich ukladaní sa neukladajú len hexa hodnoty daných znakov reťazca. Po každom uložení znaku sa ukladá aj hexa hodnota 0x00, resp. znak '\0'. Preto je potrebné násobiť dĺžku ukladaných dát číslom dva.

Iná situácia je pri typoch hodnôt ako REG_BINARY alebo REG_RESOURCE_LIST. Modifikácia dát takýchto typov hodnôt je totožná so spracovaním reťazcových typov dát. Rozdiel je v tom, že pri týchto typoch dĺžku dát nie je potrebné násobiť dvoma, nakoľko dochádza k ukladaniu dát v hexa formáte a ich zobrazovaniu v takom istom formáte.

5.4 Pridanie kľúča a pridanie hodnoty

Funkcie pridania kľúča a pridania hodnoty sú do určitej časti implementované podobne. V oboch najskôr dôjde ku kontrole, či prvok, ktorý chce užívateľ pridať už náhodou neexistuje. Ak neexistuje, funkcia vytvorí štruktúru resp. bunku daného prvku, ktorú naplní inicializačnými hodnotami a menom, ktoré zadal užívateľ. Funkcia následne vyhľadá v bufferi voľné miesto, kam túto bunku uloží.

Pri pridaní hodnoty sa rozlišujú dva prípady. Prvým je situácia, keď kľúč, ku ktorému bola hodnota pridaná, zatiaľ žiadne hodnoty nemal. V tomto prípade sa alokuje miesto pre uloženie zoznamu offsetov hodnôt a do neho sa uloží offset miesta, na ktoré bola bunka *vk* pridávanej hodnoty

uložená. Offset zoznamu hodnôt sa uloží do bunky *nk* daného kľúča a v tej istej bunke sa zvýši počítadlo hodnôt kľúča.

Ak kľúč už má nejaké hodnoty a užívateľ iba pridal ďalšiu, offset bunky *vk* tejto hodnoty sa pridá na koniec zoznamu offsetov buniek a v bunke *nk* sa zvýši počítadlo hodnôt. V prípade, ak by sa zoznam offsetov pridaním ďalšieho offsetu zväčšil natoľko, že by sa na pôvodné miesto nezmestil, funkcia pre neho nájde nové umiestnenie. Nové umiestnenie sa potom uloží do bunky *nk*.

Nová hodnota, ktorá bola pridaná ku kľúču, po týchto krokoch nebude obsahovať žiadne dáta. resp. dáta budú nulové. Preto, ak bude chcieť užívateľ k novej hodnote pridať aj nejaké konkrétne dáta, musí to urobiť pomocou funkcie modifikácie dát hodnoty.

Pri pridaní kľúča je potrebné si uvedomiť, že takisto ako hodnotu, aj kľúč možno pridať iba k nejakému kľúču. Nemožno pridať nový koreňový kľúč. Znamená to teda, že každý pridaný kľúč má nejakého predchodcu a teda je podkľúčom kľúča, ku ktorému bol pridaný. Tohto predchodcu možno nazvať aj rodičovským kľúčom.

Prvá možnosť pridania kľúča je pridať tento kľúč ku kľúču, ktorý zatiaľ nemá žiadne podkľúče. V tomto prípade, podobne ako pri hodnotách, aplikácia alokuje priestor pre zoznam offsetov podkľúčov a uloží do neho offset miesta, na ktoré bola uložená bunka *nk* pridaného kľúča. Zároveň sa v rodičovskom kľúči zvýši počítadlo podkľúčov a upraví sa informácia o offsete podkľúčov a upraví sa informácia o offsete zoznamov podkľúčov..

Pri druhej možnosti, keď už kľúč má nejaké podkľúče, je situácia zložitejšia. Ako bolo popísané v kapitole 4.3.4, offsety podkľúčov môžu byť uložené v rôznych typoch buniek. Preto aplikácia najskôr zistí, na aký typ bunky ukazuje rodičovský kľúč. Ak ide o bunku typu *lf* alebo *lh*, offset pridaného kľúča sa do nej pridá a v bunke rodičovského uzla sa aktualizuje počet podkľúčov. Hoci pri pridávaní podkľúčov v systéme Windows sa už nedodržuje ich abecedné poradie a offsety nových podkľúčov sa pridávajú jednoducho na koniec zoznamu, naša aplikácia implementuje dodržiavanie abecedného poradia aj pri novo pridaných podkľúčoch.

Ak rodičovský kľúč ukazuje na bunku typu *ri*, aplikácia postupne prehľadáva bunky *li* zisťuje kam by z abecedného hľadiska mal byť nový kľúč zaradený. Ak dané miesto nájde, podobne ako pri bunkách *lf* a *lh*, posunie nasledujúce offsety podkľúčov o potrebný počet bajtov ďalej a na uvoľnené miesto uloží offset nového podkľúča prípadne v prípade buniek *lf* a *lh* aj ďalšie štyri potrebné bajty. Pri každom z týchto troch typov buniek sa následne zvýši počítadlo podkľúčov, takisto ako aj v bunke *nk* rodičovského kľúča. Taktiež platí, že keby rozšírenie bunky spôsobilo, že ju nie je možné uložiť na jej pôvodné miesto, vyhľadá sa miesto nové. Následne sa bunka na toto miesto presunie, miesto pôvodnej bunky sa zmaže a v bunke *nk* sa upraví položka zoznamu offsetov. V prípade bunky *li* aplikácia upraví miesto umiestnenia bunky *li* v bunke *ri* na príslušnom mieste.

5.5 Vymazanie kľúča a vymazanie hodnoty

Funkcia vymazania kľúča rovnako ako funkcia vymazania hodnoty, najskôr zistí, či kľúč, ktorý chce užívateľ vymazať naozaj existuje. Ak áno, je možné pristúpiť k samotnému vymazávaniu.

V prvej etape vymazávania funkcia zisťuje, či daný kľúč má nejaké hodnoty a podkľúče. Kľúč totiž nemožno vymazať bez toho, aby neboli vymazané hodnoty a podkľúče k nemu patriace. Ak teda kľúč má nejaké hodnoty, zavolá sa funkcia pre ich vymazanie. Ďalej, ak kľúč má aj podkľúče, funkcia zo zoznamu offsetov, postupne zistí ich offsety a pre každý z nich rekurzívne zavolá funkciu vymazania kľúča. Týmto bude zabezpečené, že aj každý podkľúč kľúča, ktorý chce užívateľ vymazať, bude skontrolovaný na obsah hodnôt a podkľúčov. V prípade ich existencie budú vymazané resp. bude funkcia volaná opätovne. Postupne sa však funkcia prepracuje ku kľúču, ktorý už nemá žiadne podkľúče a spätne bude vymazávať všetky kľúče, až nakoniec vymaže podkľúče kľúča, ktorý chcel vymazať užívateľ.

Následne funkcia zistí, čí kľúč, ktorých chce užívateľ vymazať, nie je podkľúčom nejakého kľúča. Zistí to tak, že pomocou položky bunky *nk* odstraňovaného kľúča obsahujúcej offset rodičovského kľúča, sa prepracuje k tomuto kľúču a k jeho zoznamu podkľúčov. V tomto zozname následne vyhľadá vymazávaný kľúč, odstráni ho z tohto zoznamu a zníži počet podkľúčov rodičovského kľúča. V tejto chvíli neobsahuje žiadna z buniek súboru podregistrov odkaz na vymazávaný kľúč a funkcia ho môže jednoducho vymazať.

Funkcia vymazania hodnoty najskôr vyhľadá hodnotu, ktorú chce vymazať v zozname offsetov kľúča, ku ktorému hodnota patrí. Následne tento offset vymaže a zníži počet hodnôt kľúča. Funkcia však má aj naďalej poznamenaný offset danej hodnoty, keďže ho ešte potrebuje. Pomocou neho pristúpi k obsahu bunky *vk* danej hodnoty. Na základe položky obsahujúcej offset dát danej hodnoty, sa dostane na miesto dát a vymaže ich. Po tomto kroku dôjde k vymazaniu celej bunky *vk* a tým k zmazaniu hodnoty.

Ak by vymazávaná hodnota bola poslednou hodnotou, ktorú kľúč mal, funkcia v bunke kľúča zmení obsah položky ukazujúcej na zoznam offsetov hodnôt, na hodnotu 0xffffffff a počet hodnôt zníži na 0. Po tomto kroku už kľúč nebude mať žiadne hodnoty.

5.6 Problémy pri implementácii

Ako pri každej implementácii aj pri implementácii tejto aplikácie sa objavilo niekoľko problémov. Za najdôležitejšie však možno považovať dva, ktoré do značnej miery ovplyvnili funkčnosť niektorých funkcií. Tieto problémy popisuje nasledujúca časť.

5.6.1 Vymazávanie

Prvým závažným problémom bolo vymazávanie kľúčov a hodnôt. Pri implementácii funkcií súvisiacich s vymazávaním, som sa rozhodol pre postup, ktorý sa mi zdal byť logický a síce postup, ktorý bude dané prvky vymazávať ako to bolo popísané v podkapitole 5.5. Ako sa však neskôr ukázalo, napríklad pri vymazávaní hodnôt, vykonať kroky popísané v kapitole 5.5 a vymazať obsah bunky *vk* danej hodnoty, nie je postačujúce. Po takomto mazaní totiž vznikalo voľné miesto, s ktorým si nevedel operačný systém Microsoft Windows poradiť a teda systém nenaštartoval, nakoľko považoval modifikovaný súbor podregistrov za poškodený.

Keďže každý bajt v súbore podregistrov má svoj význam, pochopil som, že uvoľnené bajty sa v súbore nemôžu nachádzať len tak. Preto som implementáciu týchto funkcií rozšíril o počítanie uvoľneného miesta a jeho označovanie. Označovanie spočíva v tom, že na začiatok uvoľneného miesta bude uložená informácia o tom, aký veľký súvislý priestor sa uvoľnil. Otestovanie správnosti tejto metódy na jednoduchých príkladoch viedlo k pozitívnym výsledkom, čo sa prejavilo aj bezproblémovým štartom a funkčnosťou operačného systému Windows. Pri následnom testovaní na kľúčoch s väčším množstvom hodnôt rôzne rozmiestnených v súbore podregistrov, sa však prejavili nedostatky v implementácii. Tie boli spôsobené tým, že pri vymazávaní väčšieho množstva hodnôt kľúča v ľubovoľnom poradí, vznikalo voľné miesto vždy na inom mieste. Tým pádom bolo potrebné kontrolu tohto miesta a jeho spočítavanie prepracovať.

Bolo potrebné zabezpečiť, aby pri uvoľnení určitého miesta vzniknutého či už vymazaním bunky *vk* alebo vymazaním dát hodnoty, bolo prípadné novovzniknuté voľné miesto kontrolované pred aj za aktuálnou pozíciou v súbore. Okrem toho malo dôjsť k spojeniu novovzniknutého miesta s už existujúcim, ak sa nachádzali pri sebe a informáciu o ich veľkosti bolo potrebné aktualizovať. Tieto rozšírenia mojej metódy sa ukázali byť funkčné, avšak objavili sa ďalšie komplikácie. Tie sa týkali presunu zoznamu offsetov hodnôt. Ak sa totiž hodnoty, čiže bunky *vk*, nachádzajú medzi bunkou kľúča, ku ktorému patria, a zoznamom offsetov týchto hodnôt, tento zoznam by sa pri uvoľnení miesta mal presunúť. Čo v podstate znamená, že ak sa uvoľní miesto po vymazaní niektorej z buniek *vk*, modifikovaný zoznam offsetov by sa za určitých podmienok mal na toto miesto presunúť. Ak totiž zoznam offsetov zostane na svojom pôvodnom mieste, súbor podregistrov nebude systémom považovaný za korektný a systém Windows sa nespustí. Spôsob, ako zistiť miesto, na ktoré je potrebné zoznam offsetov hodnôt presunúť, však nie je celkom zrejmý. Neplatí pravidlo, že ak sa pred zoznamom offsetov uvoľní nejaký priestor týkajúci sa tohto zoznamu, presunie sa zoznam na toto miesto. Pravidlá ako a kedy možno zoznam offsetov hodnôt presúvať jednoznačne súvisia s pravidlami obsadzovania a správy voľného miesta v súbore podregistrov. Táto komplikácia však už súvisí s druhým problémom, ktorý bude popísaný v podkapitole 5.6.2.

Po skúmaní správania sa nástroja regedit, v súvislosti s vymazávaním hodnôt, som zistil nasledujúce odlišnosti. Nástroj regedit pri vymazávaní hodnôt, prípadne kľúčov, v skutočnosti nič

nemaže. Namiesto mazania dochádza iba k zmene štyroch bajtov popisujúcich veľkosť bunky v negatívnom tvare. Táto zmena je veľmi prostá a spočíva v prepísaní negatívneho tvaru do pozitívneho. Znamená to teda, že ak je pred bunkou jej veľkosť vyjadrená v negatívnom tvare, bunka existuje a používa sa. Ak je vyjadrená v tvare pozitívnom, bunka sa nepoužíva. Nemožno totiž povedať, že by bola vymazaná. Týmto krokom možno jednoducho získať informáciu o uvoľnenom mieste. Hoci toto miesto nie je vymazané, je za neho považované. A aj keby sa toto miesto vymazalo, nemalo by žiadnu inú veľkosť, iba veľkosť zmazanej bunky. Premena negatívneho tvaru na pozitívny teda poskytuje informáciu, ktorú som ja v mojej metóde vypočítaval. Potvrdenie pravdivosti tohto zistenia možno nájsť aj v [24].

Pri vymazaní viacerých hodnôt sa nedeje nič iné. Negatívny tvar veľkosti sa zmení na pozitívny pri každej z buniek, ktorej sa vymazávanie týka. Rozdiel je iba v tom, že ak dochádza k premene negatívneho tvaru veľkosti bunky na pozitívny pri bunke, ktorá susedí s bunkou, kde už táto zmena prebehla, tak bunka, u ktorej k zmene práve dochádza, si ku svojej veľkosti pripočíta aj veľkosť susednej bunky. Veľkosť susednej bunky však nijakým spôsobom nemodifikuje. Toto vedie k tomu, že napríklad pri vymazaní všetkých hodnôt, sa na mieste veľkosti prvej v poradí, z pohľadu uloženie v súbore, nachádza veľkosť celkového vymazaného priestoru. Pri ostatných bunkách sa nachádzajú parciálne výsledky sčítavania tejto veľkosti, prípadne pôvodné veľkosti vzniknuté premenou pozitívneho tvaru na negatívny, čo pôsobí pomerne neprehľadne a chaoticky.

Pri porovnávaní týchto dvoch metód možno dospieť k niekoľkým záverom. Metóda používaná nástrojom regedit je jednoduchšia, nakoľko pracuje vždy iba s prvými štyrmi bajtmi buniek a okrem toho, kde sa tieto bunky nachádzajú, nepotrebuje poznať iné informácie. Hoci je obsah podregistra po tejto metóde omnoho neprehľadnejší ako po mnou implementovanej metóde, je táto metóda pravdepodobne menej náchylná na chyby. Okrem toho, nakoľko metóda používaná nástrojom regedit vymazané bunky v skutočnosti nevymazáva, nevzniká tu ani potreba presunu zoznamu offsetov hodnôt. Táto nutnosť sa naopak v mojej metóde prejavila ako problém, ktorý hoci priamo nesúvisí s vymazávaním, stále ostáva problémom. Ako však vidno metóda používaná v operačných systémoch Microsoft Windows, nie je tou jedinou použiteľnou metódou.

5.6.2 Správa voľného miesta

Druhý a zároveň závažnejší problém sa týka vyhľadávania vhodného voľného miesta pre uloženie jednotlivých buniek a jeho správu. Tento problém sa dotýka funkcií pridania kľúča, pridania hodnoty a čiastočne aj modifikácie dát hodnoty. Okrem toho okrajovo zasahuje aj do funkcie vymazávania hodnôt. čo už však bolo popísané v predchádzajúcej podkapitole.

Vyhľadávanie voľného miesta metódou, ktorá bude v súbore podregistrov vyhľadávať voľné miesto s potrebnou veľkosťou na ľubovoľnom mieste v súbore, je nepoužiteľné. Dôvod je veľmi jednoduchý. Pri vyhľadávaní voľného miesta pre krátke prvky, akými sú v niektorých prípadoch

napríklad zoznam offsetov podkľúčov prípadne zoznam offsetov hodnôt, môže dôjsť k situácii, keď zdanlivo vhodné voľné miesto, je miestom nepoužiteľným. Takéto miesto sa totiž môže nachádzať vo vnútri niektorej z buniek a hoci z pohľadu veľkosti vyhovuje požiadavkám, je nepoužiteľné, pretože do bunky nemožno ukladať iné bunky alebo prvky.

Metóda implementovaná v mojej aplikácii pracuje podobným spôsobom, lenže keď v súbore nájde niektorú z buniek, preskočí ju a vo vyhľadávaní miesta pokračuje až za ňou. Týmto sa eliminuje možnosť nájdenia voľného miesta vo vnútri bunky. Táto metóda sa však pri testovaní ukázala ako nedostačujúca.

V systéme Windows sa totiž pre správu súboru podregistrov resp. jeho pamäte, používa zoznam voľných buniek. Po vytvorení nového kľúča alebo hodnoty sa vyhľadá tento zoznam pre príslušný súbor podregistrov, do ktorého sa bude kľúč resp. hodnota ukladať. Následne, na základe tohto zoznamu sa zistí, či existuje dostatočne veľká voľná bunka pre uloženie pridávaného prvku. V prípade, ak by dostatočne veľká bunka neexistovala, nástroj *Configuration Manager*, alokuje nový kôš a použije ho pre uloženie bunky. Odkaz na nevyužitú časť tohto koša pridá do zoznamu voľných buniek [23].

Ako bolo už spomenuté v kapitole 4.1, o implementáciu a správu registrov sa v systéme Windows stará nástroj s názvom *Configuration Manager*. Keďže tento nástroj pracuje s kópiou registrov vo virtuálnej pamäti, možno vysloviť názor, že správa súboru podregistrov sa v skutočnosti nedeje nad týmto súborom. Deje sa vo virtuálnej pamäti s aktuálnou verziou registrov a súbor podregistrov slúži len ako úložisko registrov. Preto by pre pochopenie správy voľného miesta registrov bolo potrebné preskúmať hlavne činnosť nástroja *Configuration Manager* a obsah virtuálnej pamäte operačného systému.

Na základe aktuálne dostupných informácií ako aj vyššie uvedeného jednoduchého popisu vyhľadávania a alokácie voľného miesta, možno len konštatovať, že ide o veľmi obmedzené zdroje informácií, na základe ktorých je veľmi obtiažne implementovať vlastnú správu pamäti registrov. Domnievam sa, že pre získanie hlbokých znalostí o tejto problematike, ktoré umožnia riešenie tohto problému a jeho implementáciu, je potrebné vykonať ďalšie o mnoho hlbšie a dôsledné skúmanie, ktoré je časovo veľmi náročné.

6 Malware a registre – experimenty

Vďaka možnosti exportovať obsah súborov podregistrov do textového súboru vo forme čitateľnej aj pre bežného človeka, možno tieto súbory využívať na experimenty. Cieľom týchto experimentov, bude odhaliť správanie sa škodlivého softwaru vzhľadom na registre operačného systému Windows, zistiť aké zmeny v nich boli vykonané a pripadne určiť, čo je úlohou týchto zmien.

Experimenty boli vykonávané na operačnom systéme Microsoft Windows XP Service Pack 1.

6.1 Priebeh experimentov

Pre vykonanie experimentu je potrebné mať k dispozícii dve verzie súborov podregistrov. Jedna verzia týchto registrov bude pochádzať zo systému neinfikovaného škodlivým softwarom a druhá naopak z infikovaného systému.

Pomocou skriptu sa vykoná export obsahu každého zo súborov podregistrov a následne, taktiež v rámci skriptu, sa vykoná porovnanie vzniknutých súborov. Výsledkom budú súbory obsahujúce rozdiely medzi jednotlivými súbormi podregistrov, ktoré budú hlavným zdrojom pre analýzu správania škodlivého softwaru.

Pred samotnou analýzou je potrebné si uvedomiť, že k zmenám v registroch nedochádza iba činnosťou škodlivého softwaru alebo činnosťou užívateľa v systéme. Množstvo hodnôt kľúčov sa mení už len samotným zapnutím prípadne vypnutím operačného systému. Spomeňme si niektoré z nich.

V súbore podregistrov s názvom *software*, sa nezávisle na inej činnosti mení napríklad hodnota *Seed* v kľúči *Microsoft\Cryptography\RNG*, ktorá slúži ako základ pre generátor náhodných čísel, ktorý je využívaný funkciami *CryptoAPI*. K ďalším hodnotám, ktoré sa neustále menia, patria hodnoty súvisiace s časom resp. obsahujúcu určitú časovú značku. K takým patria napríklad *StartTime* a *ExitTime* v kľúči *Microsoft\Windows NT\CurrentVersion\Prefetcher*, ktoré obsahujú informáciu o poslednom štarte a poslednom vypnutí systému Windows XP. Ďalej to môžu byť hodnoty *StartTimeLo, StartTimeHi, EndTimeLo, EndTimeHi* nachádzajúce sa pri viacerých kľúčoch alebo hodnoty *ProfileLoadTimeLow* a *ProfileLoadTimeHigh*, ktoré sa okrem kľúča *Microsoft\WindowsNT\CurrentVersion\Print*, kde vyjadrujú čas prihlásenia a čas odhlásenia užívateľa, nachádzajú taktiež pri viacerých iných kľúčoch.

Súbor podregistrov *system*, je taktiež často modifikovaný z dôvodu zápisu určitých informácií o čase, ako tomu bolo aj v predchádzajúcom prípade. Takto modifikovanou je napríklad hodnota *VideoInitTime* z kľúča *\Control\SessionManager\MemoryManagement\PrefetchParameters*, prípadne hodnota *ShutdownTime* z kľúča *ControlSet001\Control\Windows*. Okrem nich dochádza v systéme Windows XP k modifikácii hodnoty *ShutdownCount*, ktorá počíta počet vypnutí systému a nachádza

sa v kľúči *ControlSet001\Control\Watchdog\Display*. Ďalej je upravovaná hodnota *AppCompatCache* v kľúči *\Control\Session Manager\AppCompatibility* a hodnoty *LsaPid, Time, LogonTime*, ktoré sú uložené v kľúči *ControlSet001\Control\Lsa* alebo v jeho podkľúčoch a súvisia s procesom *lsass.exe* a bezpečnostnými mechanizmami operačného systému.

K zmenám v hodnôt nezávisle na činnosti užívateľa alebo škodlivého software dochádza aj v súbore podregistrov *SAM*, kde je v kľúči *SAM\Domains\Account\Users\000003EB* modifikovaná hodnota *F*.

Hodnôt, ktoré sú modifikované samotným operačným systémom je samozrejme viac a líšia sa v závislosti od toho, o akú verziu operačného systému ide. Ich poznanie je dôležité hlavne preto, aby ich modifikácia nebola považovaná za činnosť škodlivého softwaru.

6.2 Výsledky experimentov

Pre experimenty boli použité rôzne druhy a typy malware, ktoré však neboli konkrétne pomenované. Preto bude nasledujúca časť obsahovať iba predpokladané mená použitého malware, ktoré boli zistené na základe zhody medzi zmenami odhalenými aplikáciou a zmenami uvedenými na webových stránkach zaoberajúcich sa bezpečnosťou.

Okrem názvov odhaleného malware popisuje táto podkapitola časti registrov, ktoré daný malware modifikoval a dôvody modifikácie, prípadne čo touto modifikáciou dosiahol. Presné modifikácie a vplyv na registre sú z dôvodu rozsahu uvedené v prílohe 2.

Prvým škodlivým softwarom využitým pre experimentoch bol Win32.Sality. Tento malware vymazáva kľúče a hodnoty spadajúce pod kľúč *HKLM\System\CurrentControlSet\Control\SafeBoot*, čím zabezpečí nemožnosť spustenia systému v *Safe mode*, čiže v núdzovom režime. Spustenie v núdzovom režime totiž zamedzuje automatické spúšťanie *.exe* súborov a tým umožňuje ich odstránenie, čo samozrejme pre malware nie je vyhovujúce, preto tento režim znefunkční. Taktiež dôjde k vymazaniu kľúča *HKLM\System\CurrentControlSet\Services\ALG*, čo spôsobí blokovanie komunikačných portov alebo naopak otvorenie veľkého množstva portov, ktoré vytvoria ďalšiu slabé miesto v systéme z hľadiska bezpečnosti. Následne dochádza k pridaniu dvoch kľúčov typu *HKLM\SYSTEM\CurrentControlSet\Enum\Root\LEGACY_[niečo]*, pomocou ktorých bude zaregistrovaná služba, ktorá bude vytvorená informáciami uloženými v ďalšom novo vzniknutom kľúči *ControlSet001\Services\NdisFileServices32*. Názov *NdisFileServices32* je jedným z možných názvov ovládača jadra resp. služby, ktorý je týmto malwarom vytvorený a uložený v priečinku *%System%\drivers*. Konkrétna cesta k ovládaču aj s názvom súboru, v ktorom je uložený, je uložená v hodnote *ImagePath* spomenutého kľúča. Názov tohto súboru však nie je dôležitý. Dôležitý je názov, pod ktorým ovládač v systéme vystupuje a ten je uvedený v hodnote *DisplayName*.

Najdôležitejšou úlohou tohto ovládača je monitorovanie a filtrovanie sieťovej prevádzky pomocou služby *ipfilterdriver*. Malware vďaka tomu môže sledovať pakety v sieti a upravovať

reakcie webového prehliadača. Napríklad pri vyhľadávaní určitých kľúčových slov spojených s Win32.Sality, bude tento malware predpokladať, že zámerom užívateľa je zbaviť sa ho. Preto bude internetové stránky obsahujúce dané kľúčové slová blokovať. Avšak nie spôsobom, že zabráni ich zobrazeniu, ale tým, že bude neustále informovať o načítavaní stránky, ktorá sa však nikdy nenačíta.

Aby zmeny v registroch nebolo možné odhaliť prípadne ich opraviť, pomocou kľúča *HKCU\Software\%UserName%914\-72398023* sa zakáže spúšťanie nástroja regedit, ako aj nástroja Task manager, v ktorom by bolo možné odhaliť bežiace nežiaduce procesy. Nakoniec sa v kľúči *HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\policies\system* zakážu informačné správy a rôzne notifikácie antivírusov a firewallov a zabezpečí sa spustenie samotného Win32.Sality pri štarte systému v kľúči *HKLM\Software\Microsoft\Windows\CurrentVersion\Run* [29].

Hoci tento spôsob zabezpečenia spustenia škodlivého softwaru je najrozšírenejším spôsobom ako zabezpečiť štart malware pri štarte systému, nie je jediným spôsobom. Ďalšie existujúce a často používané spôsoby sú zobrazené v nasledujúcej tabuľke 6.1 prebratej z [30].

Tabuľka 6.1: Spôsoby automatického spustenia malware

Názov kľúča	Hodnota
HKLM\Software\Microsoft\Windows\CurrentVersion\Run	*
HKLM\System\CurrentControlSet\Service\<servicename>	ImagePath
HKCU\Software\Microsoft\Windows\CurrentVersion\Run	*
HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\SharedTaskSheduler	*
HKCU\Software\Microsoft\Windows\CurrentVersion\ShellServiceObjectDelayLoad	*
HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\Browser Helper Objects	*
HKCU\Software\Microsoft\Windows NT\CurrentVersion\Winlogon	Userinit
HKCU\Software\Microsoft\Windows\CurrentVersion\Policies\Explorer\Run	*
HKCU\Software\Microsoft\Windows NT\CurrentVersion\Winlogon\Notify\<name>	DIIName
HKCU\Software\Microsoft\Windows NT\CurrentVersion\Winlogon	Shell

Ako vyplýva z tabuľky 6.1 a z predchádzajúceho popisu malwaru Win32.Sality, je zrejmé, že aj tento samotný malware využíva viac spôsobov pre zabezpečenia automatického spustenia jeho komponent. Kým samotný malware sa spustí už spomenutým spôsobom, ovládač jadra sa spustí spôsobom využívajúcim *ImagePath*. Dokazuje to, že v tabuľke 6.1 popísané spôsoby sa naozaj používajú a možno očakávať, že sa budú vyskytovať aj v nasledujúcich experimentoch.

Ďalší z experimentov viedol opäť k zisteniu, že použitým škodlivým softwarom je Win32.Sality. Modifikované registre vyzerali takmer totožne až na malé odchýlky, ktoré je nutné spomenúť. Hlavný rozdiel bol v názve ovládača jadra. Hoci v predchádzajúcom experimente sa ovládač volal *NdisFileServices32*, v ďalšom už mal názov *dpti930*. Nie je to však náhoda, pretože

názvov ovládača jadra existuje viac. Okrem už spomenutých sa môžu objaviť aj názvy *asc3360pr, WMI_MFC_TPSHOKER_80, DAC970NT, AIC32P, abp470n5, AMSINT32, MCIDRV_2600_6_0* a iné, pričom však stále ide o malware Win32.Sality.

Ďalší z malwarov, ktorý bol využitý pri experimentovaní bol identifikovaný ako vírus Win32.Alman.B a do súborov podregistrov pridáva niektoré z už spomenutých kľúčov. Patria k ním napríklad kľúč *HKLM\SYSTEM\CurrentControlSet\Enum\Root\LEGACY_CDRALW* a kľúč *HKLM\System\CurrentControlSet\Services\cdralw* , s ktorých pomocou vírus zabezpečí automatické spustenie seba samého. Druhý zo spomenutých kľúčov bude obsahovať hodnoty, medzi ktoré bude patriť aj hodnota *ImagePath*, ktorá bude obsahovať informácie o umiestnení a názve súboru, ktorý bude spúšťaný ako služba systému Windows. Ako vidno v prílohe 2., hodnota *ImagePath* obsahuje dáta v tvare hexadecimálnych čísel. Po ich konverzii na znaky však zistíme, že v tomto konkrétnom prípade sa jedná o súbor *system32\DRIVERS\nvmini.sys*. Tento súbor však nie je jediný, ktorý je malwarom Win32.Alman.B využívaný na spomenutý účel. Okrem súboru *nvmini* môže totiž využívať aj súbory s názvom *cdralw* alebo *riodrvs*. Takisto aj názvy kľúčov prípadne hodnôt, obsahujúce v tomto prípade slovo *cdralw*, sa môžu líšiť a môžu napríklad obsahovať aj kľúčové slovo *nvmini*. Vírus môže taktiež vytvoriť kľúč *HKEY_LOCAL_MACHINE\Software\Google* [31].

Nasledujúci škodlivý software použitý pre experimentovanie používa pre svoje automatické spustenie po štarte systému najbežnejší spôsob, a to pridanie hodnôt do kľúča *HKLM\Software\Microsoft\Windows\CurrentVersion\Run*. V prípade tohto malwaru ide o dve hodnoty, ktoré vedú k automatickému spusteniu spustiteľných súborov *spools.exe* a *cftmon.exe*. Názvy týchto súborov nie sú zvolené náhodne, ale sú odvodené od skutočných programov *spoolsv.exe* a *ctfmon.exe* patriacich do operačných systémov Windows. Zatiaľ čo program *spools.exe* sa pokúša ukončiť beh antivírusových programov nainštalovaných v systéme, monitorovať užívateľovu aktivitu na sieti a jeho súkromné informácie a prípadne ich zasielať útočníkovi ako aj udržiavať v systéme zadné vrátka pre útočníka, program *cftmon.exe* zaznamenáva stlačenia klávesnice a posiela ich útočníkovi [32]. Na druhej strane program *spoolsv.exe* umožňuje tlač dokumentov bez blokovania funkčnosti systému na pozadí a program *ctfmon.exe* sleduje aktívne okná systému a poskytuje podporu služieb vstupu textu. Nejde teda o programy, ktoré by boli z hľadiska funkčnosti systému nepodstatné a ich existencia v hodnotách kľúčov registrov neprekvapuje. Preto jednoduchá zmena názvu programu spúšťaného malwarom často vedie k nepovšimnutiu si, že nejde o pôvodné programy ale o škodlivé programy. Takýto spôsob pomenovávania komponent škodlivého softwaru teda slúži ako maskovania pred odhalením.

Ďalším upravovaným kľúčom je kľúč *HKCU\Software\Classes\exefile\shell\open\command*. V ňom dôjde k prepísaniu dát hodnoty z tvaru *""%1" %*"* na tvar *"C:\Documents and Settings\%UserProfile%\cftmon.exe "%1" %*"*. Tento kľúč umožňuje definovať program, ktorý má byť vykonaný resp. spustený pri spustení ľubovoľného spustiteľného *.exe* súboru, pričom tento *.exe* súbor mu bude predaný ako parameter. Štandardne obsahuje kľúč hodnotu s dátami v tvare *""%1"*

%"*, čo prakticky znamená, že pri spustení *.exe* súboru sa nevykonáva nič okrem spusteného *.exe* súboru. Modifikáciou dát tejto hodnoty do tvaru uvedeného vyššie však dochádza pri spustení ľubovoľného *.exe* súboru aj k spustenie súboru *cftmon.exe*, ktorý je škodlivým súborom. Jednoduché vymazanie tohto súboru však problém nerieši, pretože hodnoty tohto kľúča zostanú nezmenené. To spôsobí, že pri spustení ľubovoľného *.exe* súboru bude operačný systém naďalej chcieť spustiť aj súbor *cftmon.exe*, ktorý však už existovať nebude. Jeho spustenie bude teda neúspešné, čo spôsobí to, že nebude možné spustiť žiadny *.exe* súbor. Pre odstránenie tohto problému bude potrebné opraviť hodnoty spomínaného kľúča napríklad aj pomocou nástroja regedit. Pre jeho spustenie však najskôr bude potrebné prepísať jeho príponu *.exe* na príponu *.com* [3].

Na základe úprav registrov bol tento malware identifikovaný ako trójsky kôň Troj_SmallTro.hi.

Ďalším malwarom použitým v experimentoch je trójsky kôň Trojan/Refroso.298E, ktorý vykonal nasledujúce zásahy do súborov podregistrov. Vytvoril kľúče súvisiace s vytvorením resp. udržiavaním zadných vrátok do systému, zaznamenávaním a odosielaním stlačených kláves. Konkrétne ide o kľúče *HKCU\Software\Bifrost* a *HKLM\SOFTWARE\Bifrost*. Taktiež vytvoril kľúč pre zabezpečenie automatického štartu malwaru po štarte systému spôsobom, ktorý sa síce nenachádza v tabuľke 6.1 a nevyužíva sa príliš často, ale niektoré druhy malware ho používajú. V tomto konkrétnom prípade ide o kľúč *HKLM\SOFTWARE\Microsoft\ActiveSetup\InstalledComponents\{9D71D88C-C598-4935-C5D1-43AA4DB90836}*. Ten obsahuje hodnotu *stubpath*, ktorej dáta obsahujú informácie o súbore, ktorý má byť spustený. Okrem toho dochádza k modifikácii kľúča *HKLM\Software\Microsoft\Windows\CurrentVersion\Run* pridaním hodnoty, ktorá zabezpečí automatické spustenie programu *runouce.exe*. Aj v tomto prípade ide o nepatrnú úpravu názvu programu, ktorý sa spúšťa. Súčasťou systému Windows je totiž program *runonce.exe*, kým program *runouce.exe* je škodlivý.

Pri ďalšom experimente bol použitý malware, ktorý, ako sa ukázalo po analýze jeho zmien v registroch, sa nazýva W32.Mumawow.F a často je označovaný aj ako Win32/Anilogo.f. Označovanie a pomenovávanie jednotlivých malwarov je vo všeobecnosti veľmi nejednoznačné. Spravidla existuje pre ten istý škodlivý software väčšie množstvo názvov, ktoré sa od seba často značne líšia. Dôvod je jednoduchý a je daný tým, že medzi jednotlivými výrobcami ochranného softwaru nedochádza k dohode o používaní spoločného názvu pre ten istý malware. Preto aj názvy jednotlivých malwarov uvedených v prílohe 2. a v tejto kapitole sú len niektorými z mnohých možných.

Automatické spustenie škodlivého softwaru je v prípade W32.Mumawow.F zabezpečené modifikáciou kľúča *HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run*, ktorý zabezpečí po štarte systému spustenie programu *smss.exe*. Program *smss.exe* je súčasťou systému Windows a je uložený v priečinku *C:\Windows\System32*. Ako však možno vidieť v prílohe 2., hodnota tohto kľúča

s názvom *TBMonEx* obsahuje inú cestu k programu *smss.exe*. Čo v tomto prípade znamená, že *smss.exe* nebude programom operačného systému, ale pôjde o škodlivý software.

Ďalšie zmeny v registroch spočívajú v pridaní značného množstva podkľúčov kľúču *HKLM\SOFTWARE\Microsoft\WindowsNT\CurrentVersion\ImageFileExecutionOptions*. Tie sa postarajú o zamedzenie prístupu k bežne používaným bezpečnostným a systémovým nástrojom. K ďalším zmenám patrí ešte vytvorenie kľúča *HKLM\SOFTWARE\GoogleBA* a modifikácia hodnoty kľúča *HKCU\Control Panel\Cursors*.

Takisto ako v mnohých iných prípadoch, aj v prípade vírusu W32.Mumawow existuje viacero verzií tohto malware. V jednom z ďalších experimentov sa objavila verzia tohto vírusu s názvom Win32.Mumawow.B6CA. Ako vo väčšine prípadov, platí aj v tomto, že pri rôznych verziách malware toho istého druhu, sa verzie od seba nejako značne nelíšia. Aj v tomto prípade je činnosť vírusu totožná s činnosťou predchádzajúcej verzie, až na niektoré rozdiely. Rozdiel je napríklad v názve programu, ktorý sa má automaticky spúšťať pri štarte systému a taktiež v spôsobe zablokovania prístupu k systémovým a bezpečnostným nástrojom. Okrem toho pri verzii B6CA na rozdiel od verzie F dochádza k vytvoreniu kľúča *HKLM\SOFTWARE\logogo* namiesto kľúča *HKLM\SOFTWARE\GoogleBA*.

Konkrétne a presné zmeny v jednotlivých súboroch podregistrov vykonané škodlivým softwarom použitým pri experimentovaní možno nájsť v prílohe 2. Názvy škodlivých softwarov uvedených v prílohe 2. ako aj v tejto kapitole sú len orientačné a pre ich určenie boli použité popisy malwarov uvedené na [33] [34]. Nakoľko bolo možné škodlivý software použitý v experimentoch, pomerne dobre identifikovať, možno vysloviť záver, že export registrov do textového súboru, ktorý detekciu škodlivého softwaru umožnil, prebehol v poriadku a spoľahlivo.

Pri analýze zmien v súboroch podregistrov, ktoré sú spôsobené činnosťou malware, je potrebné poznamenať, že sledovanie zmien v registroch vyššie uvedeným spôsobom nemusí byť dostačujúce pri všetkých malwaroch. Niekedy totiž dochádza k modifikácii súborov podregistrov takým spôsobom, ktorý nie je viditeľný pri exporte obsahu tohto súboru do textového súboru. Tento spôsob modifikácie sa od bežného spôsobu veľmi neodlišuje. Malý, ale pritom zásadný rozdiel je len v tom, že malware môže vytvoriť bunku určitého typu, napríklad *nk* alebo *vk*, ktorá nebude mať väzby na žiadne iné bunky. Kým v bežnom prípade má každá z buniek vytvorená prostredníctvom malware väzby na nejaký kľúč, v tomto prípade to tak nie je a to spôsobí, že sa takáto bunka neobjaví v hierarchii kľúčov a hodnôt pri exporte obsahu súboru podregistrov do textového súboru. Z toho jasne vyplýva, že metódou porovnávania existenciu takejto bunky nemožno odhaliť.

Preto implementovaná aplikácia obsahuje funkciu, ktorá porovnáva obsah pôvodného súboru podregistrov a infikovaného súboru podregistrov v binárnej forme. Táto funkcia potom odhaľuje zmeny, ku ktorým došlo po infikovaní škodlivým softwarom. Táto detekcia sa však obmedzuje iba na priestory mimo buniek *nk*, *vk*, *sk*, *hbin* a *regf*. Ak sa teda v pôvodnom aj v infikovanom súbore nachádza na určitom mieste bunka *nk*, táto bunka sa preskakuje. Naopak ak sa v pôvodnom súbore

bunka *nk* na určitom mieste vyskytuje a v infikovanom nie, došlo k zmene a tento stav je potrebné zaznamenať. Takto zaznamenané zmeny budú uvedené v súbore, ktorý bude výstupom porovnania a okrem zmien bude obsahovať aj pôvodné hodnoty zmenených bajtov ako aj offset, na ktorom ku zmene došlo. Ďalšia doplnková funkcia umožní modifikovať a upraviť zmenené bajty napríklad do pôvodného tvaru, pričom je potrebné zadať offset, od ktorého má k úprave bajtov dôjsť.

7 Kvalitatívne porovnanie riešení a možnosti rozšírenia

Kvalitatívne porovnanie navrhnutého riešenia prístupu k registrom popísaného v tejto diplomovej práci s existujúcimi riešeniami a prístupmi, je potrebné vykonať z viacerých hľadísk.

Ako bolo dokázané v kapitole 6.2, analýza vplyvov škodlivých softwarov na registre, ktorá je založená na prístupe navrhnutom v tejto diplomovej práci, prebehla úspešne. Implementovaná aplikácia je teda schopná zabezpečiť prístup k registrom nezávislý na platforme a je tiež schopná exportovať dáta z nich do zrozumiteľnej formy v podobe textových súborov. V tomto kontexte aplikácia prevyšuje nástroje popísané v kapitole 3.8 a jediným, s ňou porovnateľným nástrojom je nástroj Offline Windows Password & Registry Editor.

Modul _winreg je rovnako ako aj nástroj regedit založený na API funkciách, ktoré sú závislé na operačnom systéme Windows a ich funkčnosť na iných platformách môže byť problematická a nie je zaručene bezchybná. Okrem toho nástroj regedit je proprietárnym nástrojom systému Windows spoločnosti Microsoft, takže nemožno očakávať jeho transformáciu na platformne nezávislý nástroj. Ani jeden z týchto nástrojov teda neumožňuje prístup k registrom nezávislý na platforme.

Program Wine síce pracuje aj na unixovej platforme, ale jeho prístup k registrom je založený na práci s textovými *.reg* súbormi. So skutočnými súbormi podregistrov, ktoré sú uložené v binárnej forme, nie je software Wine v žiadnom prípade schopný pracovať.

Na druhej strane, ak pri porovnaní implementovaného riešenia s existujúcim riešeniami budeme brať do úvahy funkcie aplikácie umožňujúce prácu s registrami, budú výsledky porovnania odlišné. Z tohto hľadiska je totiž jednoznačne najlepšou aplikáciou nástroj regedit, ktorý je súčasťou operačného systému Windows. Hoci nie je platformne nezávislý, podporuje všetky funkcie, ktoré sú pre prácu s registrami potrebné. Podobne je na tom aj modul _winreg. Nepodporuje zatiaľ síce celý rozsah API funkcií pre prácu s registrami, avšak tie najdôležitejšie áno.

V porovnaní s týmito nástrojmi software Wine podporuje podobne ako nástroj regedit väčšinu funkcií pre prácu s registrami, avšak nakoľko registre ukladá opätovne len do textového *.reg* súboru, je toto porovnanie skresľujúce. Možno totiž predpokladať, že úprava textových súborov je jednoduchšia, než úprava binárnych súborov a preto možno tieto dva nástroj len ťažko porovnávať.

Aplikácia popisovaná v tejto práci umožňuje prácu s registrami nezávisle na platforme, na ktorej je spustená. Poskytuje však len základné funkcie pre prácu s registrami, ktoré z dôvodu problémov správy pamäti popísanými v kapitole 5.6.2, nefungujú dokonale.

Porovnateľným nástrojom je len nástroj Offline Windows Password & Registry Editor, ktorý umožňuje prácu s registrami a poskytuje niektoré základné funkcie pre prácu s nimi. Tento nástroj je však implementovaný ako interaktívny editor registrov. Implementuje totiž príkazový riadok, ktorý

očakáva zadanie príkazov užívateľom. Na rozdiel od tohto nástroja aplikácia implementovaná v tejto diplomovej práci môže byť veľmi jednoducho použitá ako knižnica, ktorej funkcie budú využívať iné aplikácie. Tento prístup možno považovať za lepší, keďže využitie aplikácie v rámci inej bezpečnostnej aplikácie, je pravdepodobnejšie než jej samostatné použitie.

Možno teda povedať, že z hľadiska funkčnosti je najlepším z porovnávaných nástrojov nástroj regedit a z hľadiska zabezpečenia prístupu nezávislého na platforme je to nástroj navrhnutý v rámci tejto diplomovej práce.

Problematika registrov je však zložitá a nemožno ju obsiahnuť v jednej diplomovej práci, preto nástroj implementovaný v rámci tejto práce nedosahuje z hľadiska funkčnosti kvality nástroja regedit. Táto oblasť ale ponúka niekoľko rozšírení, ktoré by určite malo zmysel preskúmať a ktorých implementácia by viedla k zlepšeniu aplikácie možno až na úroveň nástroja regedit.

K jedným z nich patrí správa pamäti súborov podregistrov a spôsob práce s ňou. Preskúmaním a vyriešením tohto problému, by bolo možné dosiahnuť bezchybnú funkčnosť všetkých funkcií implementovaných v tejto aplikácii, čo by viedlo k vytvoreniu nástroja podobného nástroju regedit. Tento nástroj by však bol platformne nezávislý, čo by bolo jeho veľkou výhodou a s najväčšou pravdepodobnosťou by našiel uplatnenie v praxi. Ďalším rozšírením tejto aplikácie by mohlo byť užívateľské rozhranie, ktoré by zjednodušilo prácu s registrami aj bežným užívateľom. Nakoľko však nepredpokladám, že by bežný užívateľ mal ambíciu spravovať registre systému Windows z iného systému, užívateľské rozhranie nezaraďujem k najdôležitejším rozšíreniam.

Jedno z ďalších rozšírení prichádzajúcich do úvahy je rozšírenie a zdokonalenie funkcie pre porovnávanie obsahov neinfikovaných a infikovaných súborov podregistrov v binárnej forme. Zdokonalenie tejto funkcie by mohlo smerovať k cieľu vytvorenia tzv. *cross-view* detekcii. Tá spočíva v porovnaní dvoch množín dát a v následnom odhalení skrytých dát. Prvú množinu dát predstavujú dáta viditeľné systému Windows resp. dáta, ktoré možno vidieť po exporte dát súborov podregistrov do textového súboru. Druhú množinu dát budú predstavovať dáta získané priamo zo štruktúr súborov podregistrov. Pôjde teda o konkrétne bunky, ktoré tieto súbory obsahujú. Porovnaním týchto dvoch metód bude možné zistiť existenciu skrytých buniek.

Najzložitejším z možných rozšírení sa javí prispôsobenie aplikácie online práci s registrami. Súčasný stav totiž predstavuje offline prístup, čo znamená, že sa pracuje so súbormi podregistrov systému, ktorý je vypnutý. Online prístup by teda naopak predstavoval prácu so súbormi registrov v bežiacom systéme. Ako však už bolo naznačené v kapitole 4.1, operačný systém nepracuje priamo so súbormi podregistrov, ale potrebné dáta má uložené vo virtuálnej pamäti. Išlo by teda zrejme o kombináciu práce s virtuálnou pamäťou a so súbormi podregistrov. S tým by súvisel aj výskum správy pamäti súborov podregistrov, čo už bolo spomínané ako potenciálne rozšírenie. Problematika online práce s registrami je však pomerne obsiahla a vyžadovala by si dlhodobý výskum a prácu na riešení tohto problému. Možno však predpokladať, že vyriešenie tohto problému by malo výrazný prínos pre oblasť ochrany počítačových systémov.

8 Záver

Cieľom tejto práce bolo vytvorenie aplikácie pre prístup k registrom operačného systému Microsoft Windows, ktorá by bola nezávislá na platforme, umožňovala by export dát registrov a ich následnú analýzu s cieľom odhalenia zmien vykonaných škodlivým softwarom v registroch.

Druhá kapitola práce sa venuje problematike škodlivého softwaru. Popisuje jeho základné druhy, ich charakteristické znaky a popisuje ukážkové vplyvy škodlivého softwaru na registre systému Windows.

Nasledujúca kapitola upriamuje pozornosť na spôsoby uloženia konfiguračných dát operačných systémov a aplikácií, pričom prikladá dôraz na popis registrov ako úložiska konfiguračných dát systému Windows. Taktiež sa venuje popisu existujúcich nástrojov schopných pracovať s registrami systému Windows.

Nakoľko problematika registrov je spojená s operačnými systémami Microsoft Windows, ktorý je proprietárny, aj postupy spojené s registrami sú proprietárne a teda verejne neznáme. To bolo dôvodom, prečo už samotnému návrhu modelu pre uloženie dát získaných z registrov a implementácii aplikácie predchádzala podrobná analýzu uloženia registrov. Vykonaním tejto analýzy, popísanej v štvrtej kapitole, boli získané poznatky o spôsobe uloženia registrov systémom Windows v binárnych súboroch podregistrov. Bolo zistené, že súbory podregistrov, v ktorých sú registre fyzicky uložené sa členia na koše. Tieto koše, z ktorých každý obsahuje hlavičku, sa ďalej členia na bunky, ktoré sa delia na rôzne druhy. Najdôležitejšími z nich sú bunky *nk* a *vk*, ktoré obsahujú informácie o kľúčoch resp. hodnotách. Podstatné sú však aj bunky, ktoré umožňujú vytvárať vzťahy medzi kľúčmi a tým celú hierarchiu kľúčov. K takým bunkám patria bunky *lf*, *lh*, *li* a *ri*. Posledným druhom bunky dotvárajúcim hierarchiu kľúčov je bunka *sk* definujúca prístupové práva ku kľúčom.

Na základe týchto znalostí bolo možné navrhnúť model pre uloženie dát získaných z registrov. Návrh tohto modelu bol dôležitý hlavne z dôvodu, že nepokrýval všetky bunky registrov, ktoré boli odhalené pri analýze. Venoval sa totiž len bunkám podstatných z hľadiska ďalšieho využitia, ktorým malo byť odhaľovanie vplyvu škodlivého softwaru na registre.

Následne bolo možné pristúpiť k implementácii samotnej aplikácie. Tá sa okrem funkcií zaoberajúcich sa načítaním, spracovaním a opätovným vytvorením binárneho súboru, venuje aj ďalšiemu množstvu funkcií. Export dát registrov do zrozumiteľnej formy, ktorú predstavuje textový súbor je samozrejmosťou. Implementované však boli aj funkcie pre základnú prácu s registrami. Patria k nim funkcie pridania kľúča, pridania hodnoty, modifikácie hodnoty, vymazania hodnoty, či vymazania kľúča. Samozrejme bolo potrebné implementovať aj množstvo pomocných funkcií ako napríklad vyhľadávanie kľúčov, či hodnôt.

Samotná implementácia so sebou priniesla určité úskalia, ktoré odhalili ďalšie dovtedy nepoznané vlastnosti registrov a ukázali, že problematika registrov je o mnoho zložitejšia, než by sa mohlo zdať na prvý pohľad.

Žiadny z problémov, ktorý sa počas implementácie objavil však neovplyvnil funkčnosť aplikácie, ktorú bolo potrebné podľa zadania zhotoviť. Mierne nedostatky ovplyvňujú iba časti, ktoré sú rozšírením pôvodného zadania. Preto možno implementáciu aplikácie považovať za úspešnú. Toto tvrdenie je podložené aj vykonanými experimentmi, ktoré ukazujú akým spôsobom môžu byť a často aj sú registre zneužívané škodlivým softwarom. Úspešnosť experimentov a schopnosť identifikácie použitého škodlivého softwaru dokazuje spoľahlivú funkčnosť aplikácie.

Táto práca mi poskytla priestor pre spoznanie jednej z mnohých oblastí týkajúcich sa bezpečnosti. Analýza uloženia registrov a vzťahov medzi jednotlivými bunkami, ktorými sú registre tvorené, mi umožnili podrobne spoznať registre, ich vlastnosti a špecifiká ako aj hrozby s nimi súvisiace.

Prínos aplikácie možno sledovať hlavne v tom, že umožňuje pristupovať k registrom, umožňuje základnú prácu s nimi, ich export a je nezávislá na platforme. Vytvára zároveň základ, ktorý by bolo možné vylepšovať o rôzne rozšírenia.

V prípade implementácie určitých rozšírení, by totiž aplikácia mohla získať na hodnote a význame. Rozšírením, ktoré by k tomu mohli viesť je napríklad implementácia správy voľného miesta v súbore podregistrov. Správna funkčnosť správy pamäti by umožnila správnu funkčnosť ďalších funkcií a postupne by sa z aplikácie mohol stať komplexný nástroj pre správu registrov, ktorý by bol navyše nezávislý na platforme. Ďalším rozšírením by mohla byť implementácia správy registrov za behu systému. Takéto rozšírenie by znamenalo značný prínos pre oblasť bezpečnosti a bezpečnostného softwaru.

Skúmanie registrov a vplyvu škodlivého softwaru na registre je veľmi zaujímavou oblasťou bezpečnosti, ktorá však, ako ukázala aj samotná práca, je značne rozsiahla. Táto oblasť si preto vyžaduje a určite aj zaslúži ďalšiu pozornosť a úsilie, ktoré by viedlo k riešeniu ďalších problémov súvisiacich s bezpečnosťou operačných systémov.

Literatúra

[1] HANÁČEK, P. *BIS: Materiály k přednáškám ve formátu PDF* [online]. 2007. [cit. 2011-05-06]. Dostupný z WWW: < https://www.fit.vutbr.cz/study/courses/BIS/private/bis.htm>.

[2] SZÖR, Péter; FERRIE, Peter Hunting For Metamorphic. In *White Paper : Symantec Security Response* [online]. [s.l.] : [s.n.], 2003 [cit. 2010-12-08]. Dostupné z WWW: <www.symantec.com>.

[3] HÁK, Igor. *Moderní počítačové viry* [online]. [s.l.], 2005. 110 s. Bakalářská práce. Univerzita Hradec Králové. Dostupné z WWW: <http://www.viry.cz/go.php?p=viry&t=clanek&id=23>.

[4] Windows Rootkit Overview. In *White Paper : Symantec Security Response* [online]. [s.l.] : [s.n.], 2005 [cit. 2010-12-08]. Dostupné z WWW: <www.symantec.com>.

[5] ZAVEC, Aleš; TOMAŽIČ, Sašo. *What it is and how to prevent it.* In *Malware and Windows environment* [online]. [s.l.] : [s.n.], 2006 [cit. 2011-05-06]. Dostupné z WWW: <www.lkn.fe.uni-lj.si/Clanki/2006/YUINFO_2006_AZ_ST.pdf>.

[6] SUKWONG, Orathai; KIM, Hyong S.; HOE, James C. *An Empirical Study of Commercial Antivirus Software Effectiveness* [online]. 2010, PP, 99, [cit. 2010-12-09]. Dostupný z WWW: <ieeexplore.ieee.org>. ISSN 0018-9162.

[7] SophosLabs. *Naked Security* [online]. April 19, 2010 [cit. 2010-12-09]. IT security blog. Dostupné z WWW: <http://nakedsecurity.sophos.com/2010/04/19/ways-dodgy-installation-windows-registry/>.

[8] *US-CERT Vulnerability Note VU#529673* [online]. 2010-11-26 [cit. 2010-12-09]. Vulnerability Note VU#529673. Dostupné z WWW: <http://www.kb.cert.org/vuls/id/529673>.

[9] *National Vulnerability Database (NVD) National Vulnerability Database (CVE- 2010-0481)* [online]. 04/14/2010 [cit. 2011-05-06]. National Cyber-Alert System. Dostupné z WWW: <http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2010-0481>.

[10] *National Vulnerability Database (NVD) National Vulnerability Database (CVE-2007-2229)* [online]. 06/12/2007 [cit. 2011-05-06]. National Cyber-Alert System. Dostupné z WWW: <http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2007-2229>.

[11] *National Vulnerability Database (NVD) National Vulnerability Database (CVE-2007-1538)* [online]. 03/20/2007 [cit. 2011-05-06]. National Cyber-Alert System. Dostupné z WWW: <http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2007-1538>.

[12] *The Registry* [online]. c2011 [cit. 2011-05-06]. Brainbell.com . Dostupné z WWW: <http://www.brainbell.com/tutors/A+/Hardware/The_Registry.htm>.

[13] PILTON, Drew. *UnArchived Articles* [online]. Apr 26, 2008 [cit. 2010-12-09]. The History of the Windows Registry. Dostupné z WWW: <http://articles.webraydian.com/article8404-The_History_of_the_Windows_Registry.html>.

[14] KOSEK, Jiří. *XML pro každého : Podrobný průvodce*. První vydání. Praha : Grada , 2000. 164 s.

[15] VERMEULEN , Sven . *Initscripts* [online]. March 2, 2011 [cit. 2011-05-07]. Gentoo Linux. Dostupné z WWW: <http://www.gentoo.org/doc/en/handbook/handbook-x86.xml?part=2&chap=4>.

[16] MALÝ, Martin. *YAML: Serializační formát pro ukládání dat* [online]. 4. 12. 2009 [cit. 2010-12-10]. Zdrojak.cz . Dostupné z WWW: <http://zdrojak.root.cz/clanky/yaml-serializacni-format-pro-ukladani-dat>.

[17] Microsoft. *Why are INI files deprecated in favor of the registry?* [online]. 26 Nov 2007 [cit. 2010-12-10]. http://blogs.msdn.com. Dostupné z WWW: <http://blogs.msdn.com/b/oldnewthing/archive/2007/11/26/6523907.aspx>.

[18] Microsoft. *Windows registry information for advanced users* [online]. February 4, 2008 [cit. 2010-12-19]. Microsoft Support. Dostupné z WWW: <http://support.microsoft.com/kb/256986>.

[19] HONEYCUTT, Jerry. *Microsoft Windows Registry Guide*. Second Edition : [s.n.], 2005. 546 s.

[20] Microsoft. *Differences between Regedit.exe and Regedt32.exe* [online]. January 19, 2007 [cit. 2010-12-18]. Microsoft Support. Dostupné z WWW: <http://support.microsoft.com/kb/141377>.

[21] Microsoft. *Registry Functions* [online]. 11/1/2010 [cit. 2010-12-18]. Msdn.microsoft.com. Dostupné z WWW: <http://msdn.microsoft.com/en-us/library/ms724875%28v=VS.85%29.aspx>.

[22] *Offline Windows Password & Registry Editor* [online]. 2009-12-01 [cit. 2011-05-15]. Dostupné z WWW: <http://pogostick.net/~pnh/ntpasswd/>.

[23] *Using the Registry and Regedit* [online]. 1997 [cit. 2010-12-11]. Winehq.org. Dostupné z WWW: <http://www.winehq.org/docs/wineusr-guide/using-regedit>.

[24] RUSSINOVICH, Mark. Inside the Registry. *Windows NT Magazine* [online]. May 01, 1999, 5195, [cit. 2011-04-01]. Dostupný z WWW: <http://www.windowsitpro.com/article/internals-and-architecture/inside-the-registry.aspx>.

[25] MORGAN, Timothy D. *The Windows NT Registry File Format*. Sentinel Chicken Networks [online]. June 9, 2009, [cit. 2011-04-02]. Dostupný z WWW: <http://www.sentinelchicken.com/research/registry_format/>.

[26] NORRIS, Peter. *THE INTERNAL STRUCTURE OF THE WINDOWS REGISTRY* [online]. Cranfield University : Cranfield University, 2009. 122 s. Diplomová práce. Cranfield University. Dostupné z WWW: <http://amnesia.gtisc.gatech.edu/~moyix/suzibandit.ltd.uk/MSc/>.

[27] SOLOMON, David; RUSSINOVICH, Mark. *Microsoft Windows Internals*. Redmond, Washington : Microsoft Press, 2005. 893 s.

[28] THOMASSEN, Jolanta. *Forensic analysis of unallocated space in Windows registry hive files* [online]. [s.l.], 04/11/ 2008. 53 s. Diplomová práce. The University of Liverpool. Dostupné z WWW: <sentinelchicken.com/data/TheWindowsNTRegistryFileFormat.pdf>.

[29] FALLIERE, Nicolas. *All-in-One Malware: An Overview of Sality* [online]. 07 May 2010 [cit. 2011-04-15]. Symantec.com. Dostupné z WWW: <http://www.symantec.com/connect/blogs/all-one-malware-overview-sality>.

[30] *Top10 malware registry launchpoints* [online]. June 8, 2007 [cit. 2011-04-15]. F-secure.com. Dostupné z WWW: <http://www.f-secure.com/weblog/archives/00001207.html>.

[31] *Vírusová encyklopédia* [online]. 2007 [cit. 2011-04-17]. Eset.eu. Dostupné z WWW: <http://www.eset.eu/virus/alman_nad_alman_b_almanahe_b_inf_almanahe_c?lng=en>.

[32] *Spools.exe - Dangerous* [online]. April 10 2011 [cit. 2011-04-17]. Greatis.com. Dostupné z WWW: <http://www.greatis.com/appdata/d/s/spools.exe.htm>.

[33] *Anchiva Systems* [online]. c2008-2011 [cit. 2011-04-17]. Dostupné z WWW: <http://www.anchiva.com/>.

[34] *ThreatExpert - Automated Threat Analysis* [online]. c2009 [cit. 2011-04-17]. Dostupné z WWW: <http://www.threatexpert.com/>.

Zoznam príloh

Príloha 1. Zoznam API funkcií pre prácu s registrami

Príloha 2. Zoznam popisovaných zmien registrov vykonaných škodlivým softwarom počas experimentovania

Príloha 3. CD so zdrojovými kódmi, testovacím skriptom a zoznamom zmien registrov vykonaných škodlivým softwarom počas experimentovania

Príloha 1. Zoznam API funkcií pre prácu s registrami [14]

Function	Description
GetSystemRegistryQuota	Retrieves the current size of the registry and the maximum size that the registry is allowed to attain on the system.
RegCloseKey	Closes a handle to the specified registry key.
RegConnectRegistry	Establishes a connection to a predefined registry handle on another computer.
RegCopyTree	Copies the specified registry key, along with its values and subkeys, to the specified destination key.
RegCreateKeyEx	Creates the specified registry key.
RegCreateKeyTransacted	Creates the specified registry key and associates it with a transaction.
RegDeleteKey	Deletes a subkey and its values.
RegDeleteKeyEx	Deletes a subkey and its values from the specified platform-specific view of the registry.
RegDeleteKeyTransacted	Deletes a subkey and its values from the specified platform-specific view of the registry as a transacted operation.
RegDeleteKeyValue	Removes the specified value from the specified registry key and subkey.
RegDeleteTree	Deletes the subkeys and values of the specified key recursively.
RegDeleteValue	Removes a named value from the specified registry key.
RegDisablePredefinedCache	Disables handle caching for the predefined registry handle for HKEY_CURRENT_USER for the current process.
RegDisablePredefinedCacheEx	Disables handle caching for all predefined registry handles for the current process.
RegDisableReflectionKey	Disables registry reflection for the specified key.
RegEnableReflectionKey	Enables registry reflection for the specified disabled key.
RegEnumKeyEx	Enumerates the subkeys of the specified open registry key.
RegEnumValue	Enumerates the values for the specified open registry key.
RegFlushKey	Writes all attributes of the specified open registry key into the registry.
RegGetKeySecurity	Retrieves a copy of the security descriptor protecting the specified open registry key.
RegGetValue	Retrieves the type and data for the specified registry value.
RegLoadKey	Creates a subkey under HKEY_USERS or HKEY_LOCAL_MACHINE and stores registration information from a specified file into that subkey.
RegLoadMUIString	Loads the specified string from the specified key and subkey.
RegNotifyChangeKeyValue	Notifies the caller about changes to the attributes or contents of a specified registry key.
RegOpenCurrentUser	Retrieves a handle to the HKEY_CURRENT_USER key for the user the current thread is impersonating.
RegOpenKeyEx	Opens the specified registry key.
RegOpenKeyTransacted	Opens the specified registry key and associates it with a transaction.
RegOpenUserClassesRoot	Retrieves a handle to the HKEY_CLASSES_ROOT key for the specified user.
RegOverridePredefKey	Maps a predefined registry key to a specified registry key.

RegQueryInfoKey	Retrieves information about the specified registry key.
RegQueryMultipleValues	Retrieves the type and data for a list of value names associated with an open registry key.
RegQueryReflectionKey	Determines whether reflection has been disabled or enabled for the specified key.
RegQueryValueEx	Retrieves the type and data for a specified value name associated with an open registry key.
RegReplaceKey	Replaces the file backing a registry key and all its subkeys with another file.
RegRestoreKey	Reads the registry information in a specified file and copies it over the specified key.
RegSaveKey	Saves the specified key and all of its subkeys and values to a new file.
RegSaveKeyEx	Saves the specified key and all of its subkeys and values to a new file. You can specify the format for the saved key or hive.
RegSetKeyValue	Sets the data for the specified value in the specified registry key and subkey.
RegSetKeySecurity	Sets the security of an open registry key.
RegSetValueEx	Sets the data and type of a specified value under a registry key.
RegUnLoadKey	Unloads the specified registry key and its subkeys from the registry.

Príloha 2. Zoznam popisovaných zmien registrov vykonaných škodlivým softwarom počas experimentovania

Popis zmien vykonaných škodlivým softwarom bude zobrazený v nasledujúcom formáte:

1. Poradie experimentu
2. Názov škodlivého softwaru – tak ako bol identifikovaný vďaka odhaleným zmenám
3. Postupne za sebou budú nasledovať jednotlivé súbory podregistrov, bude uvedený ich názov a za ním zmeny, ktoré boli v nich vykonané
4. Ak došlo k vykonaniu veľkého množstva zmien podobného charakteru, nebudú uvedené a budú nahradené troma bodkami. Kompletné zmeny budú uvedené v prílohe 3.

Experiment 1:

Win32.Sality

SYSTEM:

```
ControlSet001\Control\SafeBoot
-"AlternateShell"="cmd.exe"
-
-ControlSet001\Control\SafeBoot\Minimal
-
-ControlSet001\Control\SafeBoot\Minimal\AppMgmt
-Default="Service"
-
-ControlSet001\Control\SafeBoot\Minimal\Base
-Default="Driver Group"
...
...
...
-ControlSet001\Control\SafeBoot\Network\{36FC9E60-C465-11CF-8056-444553540000}
-Default="Universal Serial Bus controllers"
-
-ControlSet001\Control\SafeBoot\Network\{4D36E965-E325-11CE-BFC1-08002BE10318}
-Default="CD-ROM Drive"
-
-ControlSet001\Control\SafeBoot\Network\{4D36E967-E325-11CE-BFC1-08002BE10318}
-Default="DiskDrive"
...
...
...

-ControlSet001\Services\ALG
-"Type"=dword:00000010
```

```
-"Start"=dword:00000003
-"ErrorControl"=dword:00000001
-"ImagePath"=hex(2):25,00,53,00,79,00,73,00,74,00,65,00,6d,00,52,00,6f,00,6f,
-  00,74,00,25,00,5c,00,53,00,79,00,73,00,74,00,65,00,6d,00,33,00,32,00,5c,00,
-  61,00,6c,00,67,00,2e,00,65,00,78,00,65,00,00,00
-"DisplayName"="Application Layer Gateway Service"
-"ObjectName"="NT AUTHORITY\LocalService"
-"Description"="Provides support for 3rd party protocol plug-ins for Internet
Connection Sharing and the Internet Connection Firewall"
-
-ControlSet001\Services\ALG\Security
-"Security"=hex:01,00,14,80,90,00,00,00,9c,00,00,00,14,00,00,00,30,00,00,00,
-  02,00,1c,00,01,00,00,00,02,80,14,00,ff,01,0f,00,01,01,00,00,00,00,00,01,00,
-  00,00,00,02,00,60,00,04,00,00,00,00,00,14,00,fd,01,02,00,01,01,00,00,00,00,
-  00,05,12,00,00,00,00,00,18,00,ff,01,0f,00,01,02,00,00,00,00,00,05,20,00,00,
-  00,20,02,00,00,00,00,14,00,8d,01,02,00,01,01,00,00,00,00,00,05,0b,00,00,00,
-  00,00,18,00,fd,01,02,00,01,02,00,00,00,00,00,05,20,00,00,00,23,02,00,00,01,
-  01,00,00,00,00,00,05,12,00,00,00,01,01,00,00,00,00,00,05,12,00,00,00
```

→ to isté pre ControlSet002

```
+ControlSet001\Enum\Root\LEGACY_IPFILTERDRIVER
+"NextInstance"=dword:00000001
+
+ControlSet001\Enum\Root\LEGACY_IPFILTERDRIVER\0000
+"Service"="IpFilterDriver"
+"Legacy"=dword:00000001
+"ConfigFlags"=dword:00000000
+"Class"="LegacyDriver"
+"ClassGUID"="{8ECC055D-047F-11D1-A537-0000F8753ED1}"
+"DeviceDesc"="IP Traffic Filter Driver"


+ControlSet001\Enum\Root\LEGACY_NDISFILESERVICES32
+"NextInstance"=dword:00000001
+
+ControlSet001\Enum\Root\LEGACY_NDISFILESERVICES32\0000
+"Service"="NdisFileServices32"
+"Legacy"=dword:00000001
+"ConfigFlags"=dword:00000000
+"Class"="LegacyDriver"
+"ClassGUID"="{8ECC055D-047F-11D1-A537-0000F8753ED1}"
+"DeviceDesc"="NdisFileServices32"
```

```
+ControlSet001\Services\NdisFileServices32
+"Type"=dword:00000001
+"Start"=dword:00000002
+"ErrorControl"=dword:00000001
+"ImagePath"=hex(2):5c,00,3f,00,3f,00,5c,00,43,00,3a,00,5c,00,57,00,49,00,4e,
+  00,44,00,4f,00,57,00,53,00,5c,00,53,00,79,00,73,00,74,00,65,00,6d,00,33,00,
+  32,00,5c,00,64,00,72,00,69,00,76,00,65,00,72,00,73,00,5c,00,71,00,6e,00,67,
+  00,68,00,6d,00,6e,00,2e,00,73,00,79,00,73,00,00,00
+"DisplayName"="NdisFileServices32"
+
+ControlSet001\Services\NdisFileServices32\Security
+"Security"=hex:01,00,14,80,90,00,00,00,9c,00,00,00,14,00,00,00,30,00,00,00,
+  02,00,1c,00,01,00,00,00,02,80,14,00,ff,01,0f,00,01,01,00,00,00,00,00,01,00,
+  00,00,00,02,00,60,00,04,00,00,00,00,00,14,00,fd,01,02,00,01,01,00,00,00,00,
+  00,05,12,00,00,00,00,00,18,00,ff,01,0f,00,01,02,00,00,00,00,00,05,20,00,00,
+  00,20,02,00,00,00,00,14,00,8d,01,02,00,01,01,00,00,00,00,00,05,0b,00,00,00,
+  00,00,18,00,fd,01,02,00,01,02,00,00,00,00,00,05,20,00,00,00,23,02,00,00,01,
+  01,00,00,00,00,00,05,12,00,00,00,01,01,00,00,00,00,00,05,12,00,00,00
```

NTUSER.DAT:

```
+Software\testUser914\-1001785200
+"1953719668"=dword:00000009
+"-387527960"=dword:00000000
+"1566191708"=dword:00000000
+"-775055920"=dword:0000001e
+"1178663748"=dword:0000003e
+"-
1162583880"="0200687474703A2F2F707A726B2E72752F696D672F6C6F676F342E6769660068747470
3A2F2F38392E3134392E3232372E3139342F747261746174612F"
+"791135788"="272D8BF8681934CDA4F3C82C34EA7C9ECFE9DCEEEBEBBF193003FC285423C5B2B5700
31AABAB9AA682054097FAC58B82DDEC798B435431ED7ABBC380A2AEEAD12DC18E75035B46C8925947EE
EC66A97764922DA3F9F3ED2B0E43E80880EC1E1EDDF9581F647E5926C27E6F93C90285CB1B5864C5CE5
E5B312B4550EAACAA8153"
```

SOFTWARE:

```
Microsoft\Windows\CurrentVersion\policies\system
+"EnableLUA"=dword:00000000

 Microsoft\Windows\CurrentVersion\Run
+"advap32"="200804-2nd_0442.exe/r"
```

Experiment 2:

Win32.Alman.B

SYSTEM:

```
+ControlSet001\Enum\Root\LEGACY_CDRALW
+"NextInstance"=dword:00000001

+ControlSet001\Enum\Root\LEGACY_CDRALW\0000
+"Service"="cdralw"
+"Legacy"=dword:00000001
+"ConfigFlags"=dword:00000000
+"Class"="LegacyDriver"
+"ClassGUID"="{8ECC055D-047F-11D1-A537-0000F8753ED1}"
+"DeviceDesc"="cdralw"

+ControlSet001\Services\cdralw
+"Type"=dword:00000001
+"Start"=dword:00000002
+"ErrorControl"=dword:00000000
+"ImagePath"=hex(2):73,00,79,00,73,00,74,00,65,00,6d,00,33,00,32,00,5c,00,44,
  00,52,00,49,00,56,00,45,00,52,00,53,00,5c,00,6e,00,76,00,6d,00,69,00,6e,00,
  69,00,2e,00,73,00,79,00,73,00,00,00
+"DisplayName"="NVIDIA Compatible Windows Miniport Driver"
+"Tag"=dword:00000007
+"Group"="Pointer Port"

+ControlSet001\Services\cdralw\Security
+"Security"=hex:01,00,14,80,90,00,00,00,9c,00,00,00,14,00,00,00,30,00,00,00,
  02,00,1c,00,01,00,00,00,02,80,14,00,ff,01,0f,00,01,01,00,00,00,00,00,01,00,
  00,00,00,02,00,60,00,04,00,00,00,00,00,14,00,fd,01,02,00,01,01,00,00,00,00,
  00,05,12,00,00,00,00,00,18,00,ff,01,0f,00,01,02,00,00,00,00,00,05,20,00,00,
  00,20,02,00,00,00,00,14,00,8d,01,02,00,01,01,00,00,00,00,00,05,0b,00,00,00,
  00,00,18,00,fd,01,02,00,01,02,00,00,00,00,00,05,20,00,00,00,23,02,00,00,01,
  01,00,00,00,00,00,05,12,00,00,00,01,01,00,00,00,00,00,05,12,00,00,00
```

Experiment 3:

TROJ_SMALLTRO.HI

SOFTWARE:

```
Classes\exefile\shell\open\command
-Default=""%1" %*"
+Default="C:\Documents and Settings\testUser\cftmon.exe "%1" %*"

 Microsoft\Windows\CurrentVersion\Run
+"ntuser"="C:\WINDOWS\system32\drivers\spools.exe"
+"autoload"="C:\Documents and Settings\testUser\cftmon.exe"
```

Experiment 4:

Trojan/Refroso.298E

SOFTWARE:

```
+Bifrost
+"nck"=hex:ed,1b,e6,27,b9,28,d6,32,74,c3,cd,74,fa,93,5b,67

+Microsoft\Active Setup\Installed Components\{9B71D88C-C598-4935-C5D1-43AA4DB90836}
+"stubpath"=hex(2):43,00,3a,00,5c,00,57,00,49,00,4e,00,44,00,4f,00,57,00,53,
+  00,5c,00,53,00,79,00,73,00,74,00,65,00,6d,00,33,00,32,00,5c,00,77,00,69,00,
+  6e,00,64,00,6f,00,77,00,73,00,5c,00,63,00,6f,00,6e,00,66,00,69,00,67,00,2e,
+  00,65,00,78,00,65,00,20,00,73,00,00,00

 Microsoft\Windows\CurrentVersion\Run
+"Runonce"="C:\WINDOWS\System32\runouce.exe"
```

NTUSER.DAT

```
+Software\Bifrost
+"klg"=hex:01
+"plg1"=hex:ea,44,dc,02,a3,27,d7,5f,11,ad,b9,07,da,f2,35,03,2a,35,8e,58,1b,0e,
+  11,94,d4,f9,09,09,07,4b,83,ae,da,73,44,8e,cd,f9,b9,ba,23,bd,45,ed,15,a6,77,
+  af,b8,7c,04,4d,90,91,58,5f,81,3c,53,77,15,3a,53,7a,e0,ab,e6,8b,68,38,26,56,
+  bc,15,2d,84,11,9d,c9,81,6c,09,21,62,c3,8c,70,93,2c,f6,28,c5,8f,2d,ce,f6,4d,
+  44,bf,02,36,80,61,d5,45,cd,c5,b8,8a,d8,2d,eb,dc,db,fe,5c,19,c8,2b,53,2e,86,
+  ad,27,91,69,77,8f,07,dd,00,93,42,a5,79,3e,e1,28,66,c5,23,6b,ca,45,2e,65,50,
+  b7,e3,2f,2d,e7,a0,bb,05,e0,26,05,44,ba,e3,e1,73,99,6d,ee,1d,e9,e5,6a,b7,c4,
+  4d,dc,40,29,a7,ae,15,47,d8,fe,50,d8,66,fc,51,8e,a0,72,65,33,71,3d,12,ae,ec,
+  b3,0f,9c,d0,a8,11,98,3f,d2,cb,a2,57,73,4c,d8,c2,38,23,82,0c,29,05,4e,d0,78,
+  01,56,23,67,9b,2a,ff,db,59,8a,c3,ce,ef,8f,a1,a7,ff,ae,0f,93,b8,b2,55,5e,c6,
+  4d,4c,fa,4f,03,85,41,36,3a,d8,5a,8f,41
```

Experiment 5:

Win32/Anilogo.F alebo **W32.Mumawow.F**

SOFTWARE:

```
+GoogleBA
+"setup"="yes"

Microsoft\Windows\CurrentVersion\Run
+"TBMonEx"="C:\WINDOWS\Fonts\syn00-11-22-33-44\system\smss.exe"

+Microsoft\Windows NT\CurrentVersion\Image File Execution Options\360rpt.exe
+"Debugger"="net"
+
+Microsoft\Windows NT\CurrentVersion\Image File Execution Options\360Safe.exe
+"Debugger"="net"
+
+Microsoft\Windows NT\CurrentVersion\Image File Execution Options\360tray.exe
+"Debugger"="net"
+
+Microsoft\Windows NT\CurrentVersion\Image File Execution Options\ACKWIN32.EXE
+"Debugger"="net"
...
...
...
+Microsoft\Windows NT\CurrentVersion\Image File Execution Options\_AVPM.EXE
+"Debugger"="net"

+Microsoft\Windows NT\CurrentVersion\Image File Execution Options\ÐÞ¸´¹¤¾ß.exe
+"Debugger"="net"
```

NTUSER.DAT

```
Control Panel\Cursors
+"AppStarting"=""
```

Experiment 6:

Win32.Mumawow.B6CA

SOFTWARE:

```
+logogo
+"setup"="yes"

Microsoft\Windows\CurrentVersion\Run
+"TBMonEx"="C:\WINDOWS\Fonts\system\ati2evxx.exe"


+Microsoft\Windows NT\CurrentVersion\Image File Execution Options\360rpt.exe
+"Debugger"="C:\WINDOWS\Fonts\system\ati2evxx.exe"
+
+Microsoft\Windows NT\CurrentVersion\Image File Execution Options\360Safe.exe
+"Debugger"="C:\WINDOWS\Fonts\system\ati2evxx.exe"
+
+Microsoft\Windows NT\CurrentVersion\Image File Execution Options\360tray.exe
+"Debugger"="C:\WINDOWS\Fonts\system\ati2evxx.exe"
+
...
...
...
+Microsoft\Windows NT\CurrentVersion\Image File Execution Options\_AVPM.EXE
+"Debugger"="C:\WINDOWS\Fonts\system\ati2evxx.exe"

+Microsoft\Windows NT\CurrentVersion\Image File Execution Options\ÐÞ¸´¹¤¾ß.exe
+"Debugger"="C:\WINDOWS\Fonts\system\ati2evxx.exe"
```