

# **Středoškolská odborná činnost 2006/2007**

Obor 10 – elektrotechnika, elektronika, telekomunikace a technická informatika

## **Process Inspector – správce procesů**

Autor:

Martin Dráb

Gymnázium, Praha 6, Arabská 14

160 00 Praha, 4. ročník

Konzultant práce

Ing. Božena Mannová M. Math

FEL ČVUT, Praha

**Praha 2007**

Praha

Prohlašuji tímto, že jsem soutěžní práci vypracoval samostatně pod vedením Ing. Boženy Mannové a uvedl v seznamu literatury veškerou použitou literaturu a další informační zdroje včetně internetu.

V Praze dne \_\_\_\_\_

Podpis autora \_\_\_\_\_



Process  
Inspector

---

**Martin Dráb**

## **Anotace**

Process Inspector je správce úloh s rozšířenými funkcemi. Dává uživateli možnost nahlédnout do seznamu běžících procesů, služeb a ovladačů. O každém procesu zjišťuje mnoho podrobných informací a umožňuje měnit jeho vlastnosti. Kontroluje konzistenci systémových knihoven a důležitých struktur uložených v jádru operačního systému. Aplikace obsahuje i detektor skrytě běžících procesů. Nechybí možnost zaznamenávání důležitých systémových událostí, mezi které patří spuštění nového programu nebo načtení ovladače do paměti. Program lze považovat za detektor rootkitů.

Aplikace je vytvářena pro pochopení postupů a technik, které používají autoři malwaru, a nalezení způsobu, jak se proti nim bránit. Speciální pozornost je věnována rootkitům, které se pomalu stávají velkou hrozbou.

# Obsah

<b>PODPIS AUTORA</b> .....	<b>2</b>
<b>ANOTACE</b> .....	<b>4</b>
<b>OBSAH</b> .....	<b>5</b>
<b>ÚVOD</b> .....	<b>7</b>
<b>ÚVOD DO PROBLEMATIKY</b> .....	<b>7</b>
<b>ODBOBNÝ ČLÁNEK</b> .....	<b>8</b>
<b>ROZHLÉDNUTÍ</b> .....	<b>9</b>
<b>ANALÝZA</b> .....	<b>11</b>
<b>PŘÍPADY UŽITÍ</b> .....	<b>11</b>
<b>FUNKČNÍ BODY A ODHAD PRACNOSTI, ČASU A ZDROJŮ</b> .....	<b>12</b>
<b>FUNKČNÍ MODEL</b> .....	<b>14</b>
<b>NÁVRH</b> .....	<b>15</b>
<b>UŽIVATELSKÉ ROZHRAŇÍ</b> .....	<b>15</b>
<b>Hlavní formulář</b> .....	<b>15</b>
<b>Formulář „INFORMACE O PROCESU“</b> .....	<b>16</b>
<b>Formulář „INFORMACE O VLÁKNĚ“</b> .....	<b>17</b>
<b>Formulář „INFORMACE O MODULU“</b> .....	<b>17</b>
<b>Další formuláře</b> .....	<b>17</b>
<b>POPIS JEDNOTLIVÝCH MODULŮ APLIKACE</b> .....	<b>18</b>
<b>Nejnižší vrstva</b> .....	<b>18</b>
<b>Ovladače</b> .....	<b>18</b>
<b>Moduly pracující s okny</b> .....	<b>18</b>
<b>Moduly pracující s procesy, vlákny a službami</b> .....	<b>19</b>
<b>Ostatní moduly nejnižší vrstvy</b> .....	<b>19</b>
<b>MODULY STŘEDNÍ VRSTVY</b> .....	<b>19</b>
<b>MODULY SVRCHNÍ VRSTVY</b> .....	<b>21</b>
<b>IMPLEMENTACE</b> .....	<b>22</b>
<b>Požadavky na operační systém a vývojové nástroje</b> .....	<b>22</b>
<b>Řešení některých problémů</b> .....	<b>22</b>
<b>KOMUNIKACE UŽIVATELSKÉHO ROZHRAŇÍ S OVLADAČI</b> .....	<b>22</b>
<b>PŘENOS DAT V RÁMCI OVLADAČE A PROBLÉM PRIORITY</b> .....	<b>24</b>
<b>DETEKCE PŘESMĚROVÁNÍ TOKU PROGRAMU</b> .....	<b>24</b>
<b>DETEKCE SKRYTÝCH PROCESŮ</b> .....	<b>26</b>
<b>ZÍSKÁNÍ SEZNAMU OKEN CIZÍHO PROCESU</b> .....	<b>26</b>
<b>TESTOVÁNÍ</b> .....	<b>28</b>

<u>TESTY JEDNOTLIVÝCH FUNKCÍ.....</u>	<u>28</u>
<u>KŘEST OHNĚM.....</u>	<u>28</u>
<u>VÝSLEDKY.....</u>	<u>30</u>
<u>ZÁVĚR A DISKUZE.....</u>	<u>31</u>
<u>ZÁVĚR.....</u>	<u>31</u>
<u>DISKUZE.....</u>	<u>31</u>
<u>VYHLEDÁVÁNÍ SKRYTÝCH PROCESŮ.....</u>	<u>31</u>
<u>KONTROLA SYSTÉMOVÝCH KNIHOVEN.....</u>	<u>31</u>
<u>ZOBRAZOVÁNÍ OBSAHU PAMĚTI JÁDRA.....</u>	<u>32</u>
<u>KONTROLA TABULKY INTERNÍCH SLUŽEB SYSTÉMU (SSDT).....</u>	<u>32</u>
<u>PRÁCE S PROCESY.....</u>	<u>32</u>
<u>ZÁVĚR DISKUZE.....</u>	<u>32</u>
<u>POUŽITÁ LITERATURA.....</u>	<u>33</u>

# Úvod

## **Úvod do problematiky**

Osobní počítače a notebooky nám stále více a více usnadňují život. Postupně pronikají do všech možných činností. Nemusíme už chodit do banky, abychom uskutečnili bankovní převod, nepotřebujeme psací stroj pro napsání úředních dopisů a jiných dokumentů. Mnoho lidí začíná počítač používat i při telefonování. Stručně řečeno, počítače se dostávají do středu všeho dění.

Jelikož se počítačů používá při tolika různých činnostech, jejich pevné disky obsahují velmi mnoho informací, které by mohly být snadno zneužity, očitly-li by se v cizích rukou. Přirozeně tedy lidé začali vymýšlet metody, jak se k cizím datům dostat nebo jak je zničit. Tak vznikly viry. Samotné viry jsou sice úžasné programy a dobře slouží svému účelu, ale mají jednu nevýhodu – je snadné je odhalit. Tato skutečnost hrála v neprospěch tvůrců virů, protože antiviry byly schopny detekovat jakéhokoliv záškodníka staršího data. Tvůrci virů museli odpovědět. Touto odpovědí byl vynález rootkitů – technologie, která může ochránit virus (nebo jiný nevídaný program) před odhalením.

Tím se dostáváme k definici pojmu rootkit. Jako rootkit můžeme označit program, který se snaží maximálně utajit nejen svoji přítomnost v systému, ale i přítomnost jiných programů. Někteří lidé tvrdí, že rootkity jsou ve své podstatě neškodné a záleží na tom, k čemu jsou využívány. Bohužel se rootkitů využívá hlavně na ochranu proti odhalení.

Rootkity lze rozdělit do dvou kategorií: na rootkity běžící v módu Ring 3 a na rootkity běžící v módu Ring 0. Detekovat první skupinu není příliš obtížné. Běží totiž ve stejném módu jako běžné programy (webový prohlížeč, správce souborů, kancelářský software), nemají přímý přístup k jádru operačního systému – ochrana, kterou poskytují sobě a dalším programům je tedy nedokonalá. V módu Ring 0 může běžet jen speciální typ programů – ovladače. Ovladače se zavádějí do samotného jádra operačního systému a mohou je modifikovat, jak uznají za vhodné. Detekovat rootkit běžící v módu Ring 0 může být velmi obtížné, ne-li téměř nemožné. Obě skupiny rootkitů používají různých postupů a technik, aby dosáhly svého cíle.

Společnosti zabývající se bezpečnostním softwarem odpověděly vytvořením programů, jejichž jediným cílem je vyhledávání a ničení rootkitů. Schopnosti těchto programů však nejsou dostatečné, protože nedokáží odhalit moderní rootkity. V oblasti antirootkitových nástrojů je podle mého názoru nejlepší důvěřovat freewaru, který je vyvíjen nadšenci, kteří zároveň moderní rootkity píšou.

Nedostatek kvalitního softwaru na tomto poli mne vedl k nápadu vytvořit program, který dokáže nahradit Správce úloh ve Windows a zároveň si poradí i s vyhledáváním rootkitů. Vývoj programu bude pravděpodobně velmi dlouhý a náročný, protože problematika rootkitů je velmi složitá a nedokumentovaná, s největší pravděpodobností nebude ukončen v době publikování této práce.

## Odborný článek

Program je určen převážně pro pokročilejší uživatele, kteří mají základní povědomí o operačním systému Windows a jeho architektuře. Těmto uživatelům může pomoci detekovat rootkity či jiný malware. Běžným uživatelům poslouží stejně jako Správce úloh, dokonce je možné Správce úloh touto aplikací nahradit.

Aplikace zobrazuje seznam právě běžících procesů. U každého procesu je vypsané několik základních informací (priorita, název souboru, počet vláken, rodič). Některé z těchto údajů lze měnit (priorita). Procesy můžeme ukončit či pozastavit. O každém procesu se zjišťuje mnoho dalších informací, které můžeme rozdělit do několika kategorií:

- **Obecné informace** zahrnují základní charakteristiky procesu. Jedná se o PID (jedinečný identifikátor procesu v rámci operačního systému), prioritu, počet vláken (podprocesů) a paměťové a časové statistiky.
- **Seznam modulů**, které proces používá. O každém modulu program zobrazí základní informace (název, cesta k souboru, velikost, umístění v paměti procesu). Do procesu můžeme nové moduly přidávat nebo staré odstraňovat. Process Inspector sbírá o každém modulu podrobné informace.
- **Vlákna**. Pod pojmem vlákno si můžeme představit objekt, který je zodpovědný za vykonávání kódu procesu. Samotný proces obsahuje jen prostředky, které všechna jeho vlákna sdílí. Takovým prostředkem je například jeden adresový prostor, ve kterém všechna vlákna běží. Každé vlákno je charakterizováno jedinečným identifikátorem (TID). Process Inspector zobrazuje informace o vláknech, které vykonávají kód procesu. Některé vlastnosti vláken lze měnit – můžeme zvýšit či snížit prioritu nebo určité vlákno pozastavit či ukončit.
- **Okna**. Většina aplikací běžících po Windows obsahuje uživatelské rozhraní, které je založeno na oknech. Operační systém chápe jednotlivé prvky rozhraní stejně – jako okna. Pro OS je oknem i každé tlačítko, textové pole či rolovací nabídka. Okna samozřejmě nemusí být viditelná. Každé okno je opět charakterizováno číselným identifikátorem, který však není jedinečný v rámci celého systému. Dále můžeme získat typ (textové pole, tlačítko) a text v oknu obsažený (obsah textového pole, název tlačítka).
- **Objekty jádra**. Informace o některých objektech jsou uloženy v jádře operačního systému. Patří sem procesy, vlákna, otevřené soubory, klíče registru a mnoho dalších (objekty pro synchronizaci vláken, sdílená paměť...). Těmto objektům se říká *objekty jádra*. Informace o objektech jádra, ke kterým má proces přístup, nám mohou leccos prozradit.
- **Oblasti paměti**. Každý proces má k dispozici 2 GB virtuálního adresového prostoru – to znamená, že má k dispozici až 2 GB virtuální paměti. Málokterý program tolik paměti využije – většinou volná paměť převládá nad alokovanou a rezervovanou. Program alokuje, rezervuje a uvolňuje paměť, jak právě potřebuje. V adresovém prostoru jsou tedy úseky alokované paměti různě rozesety. Každý úsek má definovaná přístupová práva, která určují, zda tam lze ukládat data nebo spouštět kód. Přístupová práva lze samozřejmě změnit, ale nedoporučuje se to.
- **Práva**. V rodině operačních systémů Windows NT není žádnému procesu dovoleno vše. Pokud chce program například vypnout počítač, musí mít na to právo. Každý program při svém spuštění dostane určitá práva, záleží na uživateli, který jej spustil.

Process Inspector také umožňuje prohlížet některá místa, kde se často mohou skrývat rootkity a další nevídané programy. Uživatel si také může nastavit procesy, kterým bude dovoleno vykonávat svou činnost – ostatní budou ukončeny nebo pozastaveny. Uživatel je také upozorněn, využívá-li nějaký proces neznámý modul.

Mnoho informací je také zjišťováno o službách nainstalovaných na počítači a o ovladačích právě běžících v jádře operačního systému. Důležité události (vytvoření nového procesu, zavedení nového ovladače do paměti) jsou monitorovány a zaznamenávány. Nechybí ani možnost detekce skrytě běžících procesů.

## **Rozhlédnutí**

Existuje mnoho aplikací, které mají podobné cíle jako budoucí Process Inspector. Mnoho z nich je freeware a přitom velmi kvalitních. V této stati se zaměřím na několik docela známých produktů, o kterých možná slyšeli i uživatelé, kteří se o počítače a operační systémy příliš nezajímají. Odkazy na níže zmíněné produkty se totiž čas od času objevují i v počítačových časopisech.

Prvním programem, o kterém bych se chtěl zmínit, je produkt firmy SysInternals **Process Explorer**. Jedná se de facto o zdokonalenou verzi Správce úloh. Process Explorer zobrazuje seznam právě běžících procesů. O každém procesu však zjišťuje mnohem více informací než Správce. Neznalý uživatel může být množstvím informací zahlcen, ale ti, kteří se trochu orientují, budou spokojeni. Program také umožňuje procesy pozastavit (to Správce neumí), což je užitečné hlavně při boji s viry a dalšími „neukončitelnými“ programy.

Další zajímavou aplikací je utilita **IceSword** od čínských vývojářů. Program zobrazuje obsah oblastí operačního systému, kde se mohou zabydlet rootkity, viry a jiný malware. Takovou oblastí je například tabulka interních služeb systému (Systém Service Descriptor Table - SSDT). Přítomnost nežádoucího softwaru může být také odhalena při prohlížení seznamu spojení. Dále je možno zjistit, které programy monitorují činnost myši a klávesnice. IceSword dokonce umožňuje vyhledávání skrytých procesů a podezřelých ovladačů. Pokud se v systému nachází skrytý proces nebo neviditelný ovladač, znamená to přítomnost rootkitu. Program pravděpodobně nezkušeného uživatele zmate, ale pro administrátora či programátora může být dobrým pomocníkem. Na závěr řeknu snad jen to, že možnosti tohoto programu jsou ještě daleko větší, než jsem zde popsal.

Produkt **RootkitRevealer** od firmy SysInternals se také zaměřuje na vyhledávání rootkitů, jak již plyne z jeho názvu. Metoda detekce je jednoduchá – program se snaží zjistit informace o všech souborech na disku několika způsoby. Jedním ze způsobů je použití funkcí poskytovaných rozhraním Windows API. O druhé metodě toho autoři moc nepíší (což je pochopitelné), ale myslím, že se jedná o čtení dat přímo z jednotlivých clusterů pevného disku. Pokud se data získaná oběma způsoby neshodují, může (ale nemusí) to být známka přítomnosti rootkitu. Podobné metody program aplikuje také na registr Windows.

Další zajímavostí RootkitRevealeru je bezesporu fakt, že jej v seznamu běžících procesů nevidíte pod pravým jménem. Samotný program se totiž spouští z náhodně pojmenované kopie. Rootkity po nějaké době začaly tento program hledat v seznamu běžících

aplikací, aby mohly zabránit odhalení. Proto bylo nutné přijmout výše popsané obranné opatření.

RootkitRevealer není určen pro uživatele-začátečníky. Jeho „nálezy“ ještě nemusí znamenat přítomnost rootkitu v systému. Ale to musí uživatel posoudit sám.

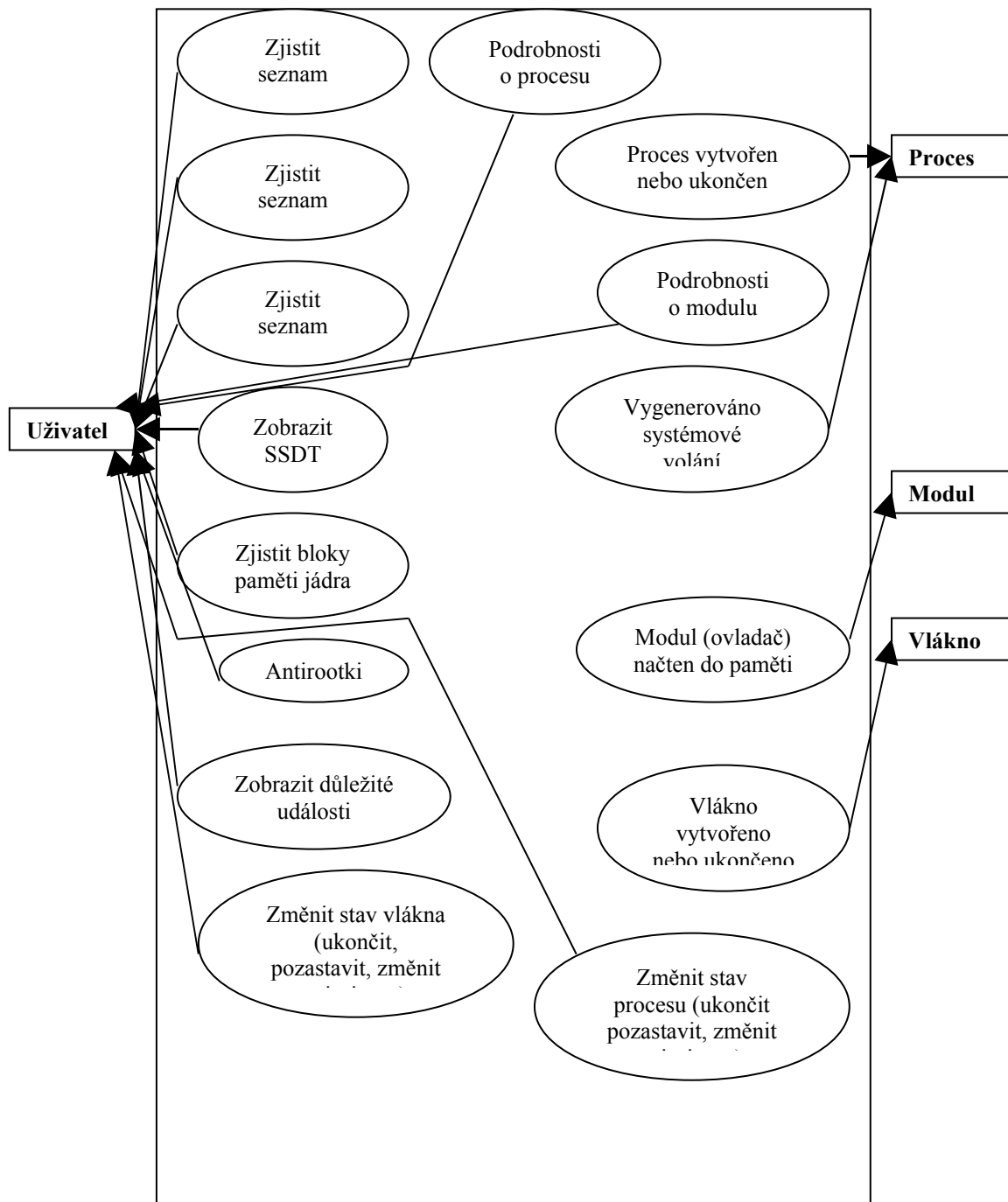
**RootkitUnhooker** je freeware, jehož jediným cílem je zbavit váš počítač rootkitů. Vyhledává skryté procesy a soubory a kontroluje konzistenci systémových knihoven a ovladačů. Autoři jsou odborníky na slovo vzatými, protože se problematikou rootkitů dlouho zabývají. Díky tomu RootkitUnhooker využívá velmi pokročilých technik a postupů, které mu podle některých zajišťují první místo mezi detektory rootkitů.

**F-Secure BlackLight** je poslední produkt, o kterém se zmíním. Cílem tohoto programu je nalezení, rozpoznání a odstranění rootkitů. Program je vhodný i pro normální uživatele, protože žádné odborné znalosti nejsou potřeba. Aplikace se pokusí nalézt skryté procesy, soubory a další objekty. Pokud je odhalen a rozpoznán rootkit, stačí kliknout na jediné tlačítko a BlackLight se nežádoucí program pokusí odstranit. Myslím, že možnosti této aplikace jsou zatím dosti omezené, ale doufám, že vývoj bude pokračovat.

Více „konkurenčních“ programů není nutné uvádět. Z jejich popisu je zřejmé, co všechno bych rád do Process Inspectoru zakomponoval. Jak se zdá, nebude toho málo.

# Analýza

## Případy užití



Obrázek 1: Případy užití

## **Funkční body a odhad pracnosti, času a zdrojů**

<b>Modul</b>	<b>Popis</b>	<b>LOC</b>	<b>Obtížnost</b>
PPROC.DLL	Nízkoúrovňové funkce pro práci s procesy (ukončení, pozastavení, spuštění...)	450	Nízká
TPROC.DLL	Nízkoúrovňové funkce pro práci s vlákny (zjištění, ukončení, pozastavení...)	100	Nízká
ModuleOperation.dll	Nízkoúrovňové funkce pro práci s DLL knihovnami (zjištění DLL používaných procesem, DLL injection, odstranění knihovny z procesu)	450	Nízká
SPROC.DLL	Nízkoúrovňové funkce pro práci se službami (services)	100	Střední
GMEM.SYS	Ovladač jádra, který umožní číst a zapisovat paměť běžně obyčejným programům nedostupnou	150	Vysoká
LOGPTM.SYS	Ovladač jádra, který bude detekovat vytvoření nového procesu, vlákna nebo načtení DLL knihovny do paměti	150	Vysoká
Win9xFunctions	Nízkoúrovňové rozhraní pro Windows 9x – zajišťuje částečnou zpětnou kompatibilitu.	360	Střední – vysoká
ClassProcesy	Rozhraní pro práci s procesy běžícími v systému (zapouzdřuje PPROC.DLL)	450	Nízká
ClassModuly	Rozhraní pro práci s DLL knihovnami používanými daným procesem (zapouzdřuje ModuleOperation.dll)	220	Nízká
ClassServices	Práce se službami nainstalovanými na PC (zapouzdřuje SPROC.DLL)	260	Nízká
ClassExports	Zjištění exportovaných funkcí DLL knihovny	180	Nízká
ClassImporty	Zjištění importovaných funkcí DLL knihovny	250	Střední
ClassDrivery	Zjišťování seznamu zavedených ovladačů	150	Nízká
ClassVlakna	Práce s vlákny, které tvoří daný proces (zapouzdřuje TPROC.DLL)	250	Nízká
ClassMemoryPages	Zjišťování seznamu paměťových stránek, které proces používá	300	Nízká

GMEM.DLL	Komunikace s ovladačem GMEM.SYS	150	Střední
LOGPTM.DLL	Komunikace s ovladačem LOGPTM.SYS	300	Vysoká
SSDT.DLL	Nízkoúrovňové funkce pro načtení a změnu SSDT	300	Střední
ClassSSDT	Rozhraní pro práci s SSDT (zapouzdřuje SSDT.DLL)	250	Nízká
Handles.DLL	Nízkoúrovňové funkce pro zjištění objektů (souborů, registrových klíčů) používaných určitým procesem	350	Střední
ProcessSecurity	Zabránění spuštění určitému procesu (lze nastavit podle několika kritérií)	500	Střední
General.DLL	Užitečné nízkoúrovňové funkce	200	Střední
MHOOKS.DLL	Nízkoúrovňové funkce pro zjištění nainstalovaných háků Windows	200	Vysoká
ClassMHooks	Zapouzdření MHOOKS.DLL	250	Nízká
ANTIR.DLL	„Anti-rootkit“ kontrola systému	500	Vysoká
MF.DLL	Zobrazení obsahu souboru nebo úseku paměti	300	Nízká
PROTECT.DLL	Ochrana Process Inspectoru proti nežádoucímu softwaru	500	Vysoká
ClassPrivs	Rozhraní pro zjištění práv určitého procesu	250	Nízká
ClassWindows	Rozhraní pro zjištění počtu a typu oken určitého procesu	250	Nízká
GUI	Grafické rozhraní aplikace	1500	Nízká
<b>Celkem LOC</b>		<b>9560</b>	
<b>Průměrná obtížnost</b>			<b>Střední (přechodný mód)</b>

*Tabulka 1: Seznam funkčních bodů (modulů) projektu*

Konstanta <sup>1</sup>	Hodnota
a	3,0
b	1,12
c	2,5
d	0,35

*Tabulka 2: Hodnoty konstant použitých ve výpočtu*

### Výpočet pracnosti, času a nákladů

Pracnost  $E = a * (KLOC)^b = 3 * 9,6^{1,12} = 38,13$  mm

Časová náročnost  $D = c * (E)^d = 2,5 * 38,13^{0,35} = 8,9$  měsíců

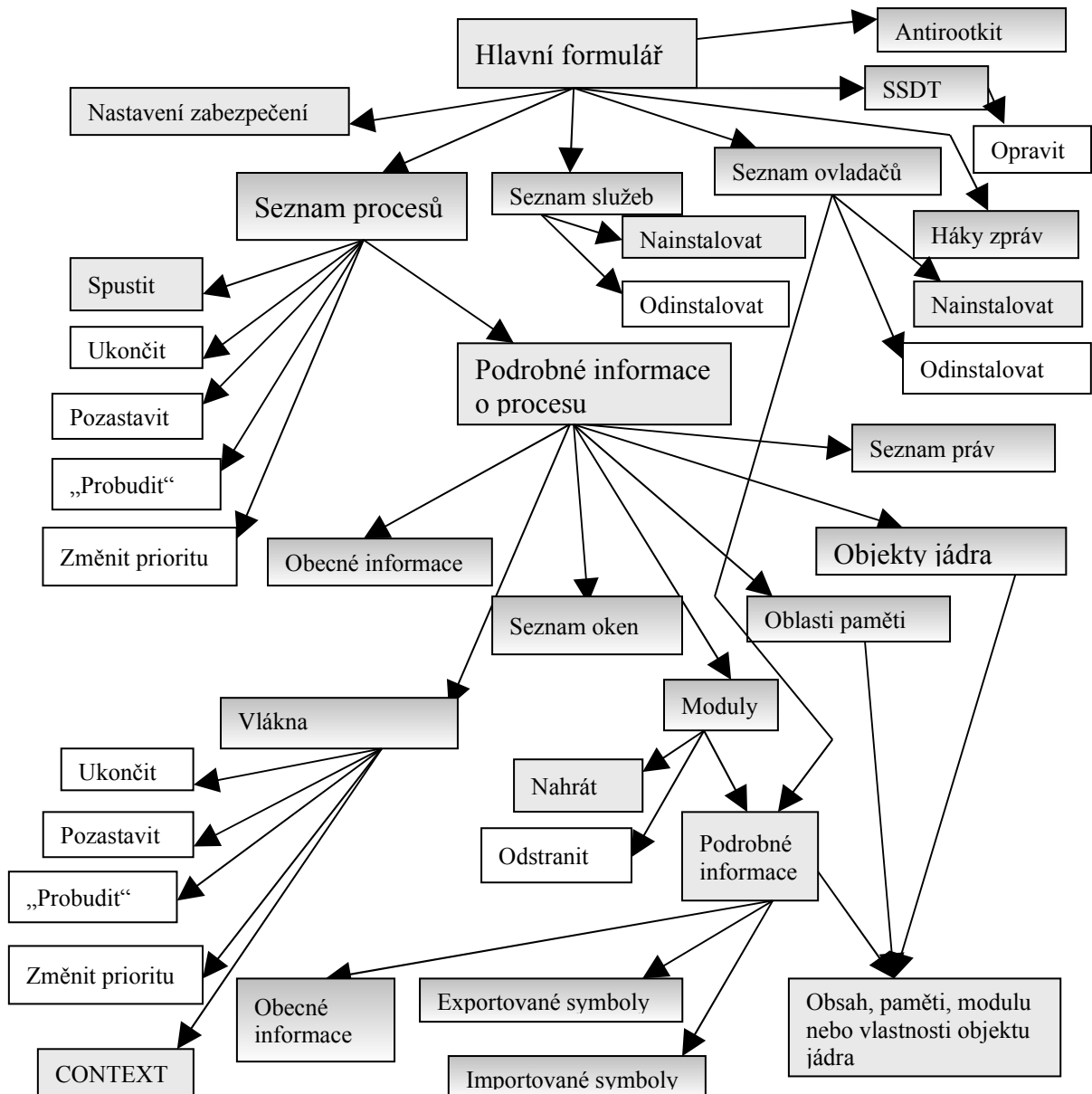
Doporučená velikost týmu  $N = E / D = 38,13 / 8,9 = 4$  lidé

Pokud počítáme náklady na jednu osobu 30 000 Kč měsíčně, celkové náklady se vyšplhají na 1 068 000 Kč.

<sup>1</sup> MANNOVÁ, B. a VOSÁTKA, K. *Řízení softwarových projektů*. Nakladatelství ČVUT, 2005. 5. 4. 3. Model COCOMO (COConstructive COSt, MOdel), s. 50 - 53

## Funkční model

**Vysvětlivky:** Světle šedou barvou jsou vyznačeny formuláře. Záložky na formulářích značí barevný přechod mezi šedou a bílou. Funkce programu, které nemají vlastní formulář nebo záložku (volí se pouze pomocí menu) jsou vyznačeny bíle.



**Obrázek 2:** Funkční model programu Process Inspector

# Návrh

## *Uživatelské rozhraní*

Uživatelské rozhraní nebude oplývat žádnými speciálními efekty, u tohoto druhu programů se spíše klade důraz na funkčnost než na efektní GUI. Protože aplikace v sobě zahrnuje mnoho funkcí, není možné pro každou z nich vytvořit samostatný formulář – uživatelské rozhraní by se stalo velmi nepřehledným. Proto jsem se rozhodl umístit většinu funkcí na několik formulářů, které budou dále rozděleny na menší celky pomocí záložek. Podívejme se tedy, jaké formuláře bude program obsahovat.

## Hlavní formulář

Hlavní formulář se zobrazí hned po startu programu. Je rozdělen sedmi záložkami a poskytuje uživateli přístup k nejdůležitějším funkcím programu. Jednotlivé funkce jsou dostupné buď z položek hlavního menu nebo z kontextového menu, které se zobrazí při kliknutí pravým tlačítkem myši na obsah určité záložky. Následuje podrobnější popis jednotlivých záložek:

- **Procesy**  
Zde se zobrazuje seznam běžících procesů. Zobrazované informace o každém procesu lze upravovat pomocí položky *Zobrazit* v hlavním menu. Všechny funkce týkající se procesů jsou dostupné v hlavním menu přes položku *Procesy* nebo přes kontextové menu.
- **Služby**  
Zpřístupňuje všechny funkce pro práci se seznamem nainstalovaných služeb. Služba je zvláštní typ procesu, který s uživatelem nekomunikuje přes standardní prostředky GUI. Služby běží nezávisle na právě přihlášeném uživateli. Všechny funkce jsou opět dostupné přes položku *Služby* v hlavním menu nebo přes kontextovou nabídku.
- **Ovladače**  
Zobrazuje seznam aktuálně běžících ovladačů. Ovladače jsou programy běžící v režimu jádra. Více informací naleznete v příloze A.
- **SSDT**  
Odtud jsou dostupné všechny funkce pro práci s tabulkou interních služeb operačního systému. Pokud je s touto tabulkou něco v nepořádku, uživatel je na to upozorněn.
- **Paměť jádra**  
Zobrazuje bloky paměti, kde se nachází jádro operačního systému. Celý obsah paměti jádra je také možné uložit na disk k dalším bezpečnostním analýzám.
- **Antirootkit**  
Pokud uživatel klepne na tuto záložku, program se pokusí nalézt nesrovnalosti v některých důležitých strukturách operačního systému a upozorní na ně. Jedná se převážně o hledání skrytých běžících aplikací a detekci změn obsahu systémových modulů. Tato „prohlídka“ systému může trvat i několik minut.
- **Události**

Program sám od sebe zaznamenává některé důležité události, které jsou zobrazeny po kliknutí na tuto záložku. Aplikace může zaznamenávat tyto události:

- Vytvoření nového procesu, nebo ukončení běžícího procesu
- Zavedení modulu (DLL knihovny) do adresového procesoru procesu, nebo zavedení nového ovladače do jádra operačního systému
- Vytvoření nebo ukončení vlákna procesu
- Volání služeb operačního systému. Kdykoliv některý z běžících procesů potřebuje například otevřít soubor, musí zavolat určitou službu jádra. Toto volání je zachyceno a zaznamenáno Process Inspektorem. Zaznamenávání událostí tohoto typu je standardně vypnuto, protože může zpomalovat chod operačního systému, doporučuji je zapínat jen na krátkou dobu.

## Formulář „Informace o procesu“

Pokud se uživatel chce dozvědět více informací o určitém procesu, označí jej a v menu *Procesy* (nebo v kontextové nabídce) klepne na položku *Podrobnosti*. Zobrazí se formulář s titulkem „Informace o procesu“. Jedná se opět o velmi rozsáhlý formulář – nachází se na něm sedm záložek. O každém běžícím programu totiž Process Inspector zjišťuje mnoho různorodých informací. Následuje stručný popis jednotlivých záložek:

- **Obecné**  
Na této záložce jsou vypsány základní charakteristiky procesu. Jedná se například o název, identifikační číslo, rodiče, prioritu a paměťové charakteristiky.
- **Moduly**  
Ukazuje seznam modulů nahrených do adresového prostoru procesu. Moduly lze z adresového prostoru procesu uvolňovat, nebo nahrávat nové. O každém modulu si uživatel může zobrazit podrobnější údaje.
- **Vlákna**  
Zobrazuje seznam vláken procesu. Jednotlivá vlákna je možné pozastavovat, měnit jim prioritu a ukončovat je. Zvědavý uživatel si může nechat zobrazit podrobnější informace o každém z nich.
- **Okna**  
Na této záložce jsou zobrazena okna, která procesu patří. Okno je základní prvek grafického uživatelského rozhraní a každá aplikace toto rozhraní využívající nějaká okna vlastní.
- **Objekty jádra**  
Pokud uživatel chce zjistit, se kterými soubory program právě pracuje, tato záložka mu poskytne odpověď; zobrazuje všechny objekty jádra, ke kterým má právě proces přístup. Je dokonce možné přecíst si obsah některých souborů.
- **Oblasti paměti**  
Zobrazuje stav virtuálního adresového prostoru procesu. Uživatel může zjistit, které bloky jsou alokované a které volné. Může si zobrazit obsah určitého bloku nebo jej uložit do souboru.
- **Práva**  
Ne každému programu je dovoleno dělat, co si zamane. Každý proces disponuje určitými právy, která mu dovolují provádět některé úkony. Práva procesu jsou odvozena od práv uživatele, pod kterým byl spuštěn, ale mohou se dokonce lišit

vlákno od vlákna. Z této záložky se uživatel dozví, která práva má proces povolena a která ne.

## Formulář „Informace o vlákně“

Tento formulář má pouze tři záložky a zobrazuje některé informace o určitém vlákně. První záložka s názvem *Obecné* obsahuje základní informace. Mezi ně patří identifikační číslo, priorita, čas vytvoření aj. V záložce *CONTEXT* se nachází výpis hodnot důležitých položek struktury *CONTEXT*. Struktura *CONTEXT* určuje hodnoty, kterými se naplní registry, když vlákno dostane čas na procesoru. Třetí záložka zobrazuje, jakými právy vlákno disponuje. Soubor práv vlákna se ve většině případů neliší od práv procesu, v jehož kontextu vlákno vykonává svůj kód.

## Formulář „informace o modulu“

Zde je v pěti záložkách pro uživatele shromážděno mnoho informací o uživatelem zadaném modulu (modulem je zde míněn libovolný PE soubor, který je právě systémem využíván – může jít tedy o DLL knihovny, EXE soubory běžících procesů i ovladače). Pro většinu uživatelů jsou tyto informace naprosto zbytečné, systémové programátory potěší.

Záložka *Obecné* opět shromažďuje základní informace o modulu (adresu v paměti, proces, který jej využívá, velikost v bytech, název souboru).

Záložka *Exporty* zobrazí seznam rutin, které modul exportuje – umožňuje ostatním PE souborům, aby je využívaly. Zvědavý programátor si může vypsát kód zvolené rutiny v jazyce assembler.

Záložka *Importy* obsahuje seznam externích rutin, které modul potřebuje pro svůj běh. U každé rutiny je samozřejmě zobrazen také název souboru, ve kterém je uložena. Pohledem na seznam importovaných rutin lze někdy odhadnout, jaký úkol má modul plnit (to se týká zejména ovladačů).

Záložka s názvem *Paměť* zobrazí obsah modulu podobně jako hexadecimální editor. Opět je možné použít interní disassembler.

V poslední záložce (*Řetězce*) se zobrazí seznam řetězců, které se v modulu nacházejí. Jedná se převážně o řetězcové konstanty, které byly definovány při jeho programování. Tyto informace nám mohou mnohé prozradit (například v jakém vývojovém prostředí byl modul zkompileován).

## Další formuláře

Process Inspector dále obsahuje několik „malých“ formulářů, u kterých ještě nebylo rozhodnuto, jak budou přesně vypadat. Tudiž jen stručně popíšu funkce, které uživateli zprostředkují.

Pomocí formuláře s názvem *Nastavení* bude možné měnit konfiguraci některých funkcí programu. Všechna nastavení budou soustředěna na jednom jediném formuláři.

Formulář „informace o objektu jádra“ zobrazí některé podrobnosti například o procesem otevřeném souboru nebo právě používaném klíči registru Windows.

Formulář *Disassembler* zpřístupňuje funkce jednoduchého disassembleru zabudovaného v programu. Disassembler je možné použít ke čtení kódu podezřelých modulů

či rutin, nebo ke studování chování nedokumentovaných podprogramů exportovaných jádrem operačního systému.

## **Popis jednotlivých modulů aplikace**

Kód programu je rozdělen do velkého počtu souborů – modulů. Moduly tvoří třívrstvou strukturu, ale hranice mezi střední a svrchní vrstvou není úplně ostrá. Nejnižší vrstva je tvořena téměř výhradně DLL knihovnamí a ovladači. Výhoda DLL knihoven je, že při úpravě kódu není nutné překompilovat celý projekt, ale jen určitou knihovnu<sup>2</sup>. Tato vrstva zprostředkovává komunikaci mezi standardním rozhraním Windows API a střední vrstvou. Díky této vrstvě lze program snadno učinit přenositelným mezi různými verzemi operačního systému Windows, pokud by se mi podařilo nalézt potřebnou dokumentaci<sup>3</sup>.

Střední vrstvu tvoří jednotky obsahující třídy. Třídy v sobě zapouzdřují funkce poskytované nižší vrstvou modulů a usnadňují programování uživatelského rozhraní. Zde se také nachází moduly některých formulářů.

Horní vrstva zahrnuje hlavní část uživatelského rozhraní a hlavní soubor programu.

Nyní se podíváme na jednotlivé vrstvy podrobněji.

## **Nejnižší vrstva**

### **Ovladače**

Ovladače činí z programu velmi mocný nástroj, protože mu poskytují naprostý přístup k jádru operačního systému. Díky nim může program nejen číst důležité struktury v chráněné paměti, ale také je měnit a tím odstranit nebezpečí škodlivého softwaru. Aplikace využívá služeb dvou ovladačů: gmem.sys a logptm.sys. Rozhraní je uloženo v DLL knihovnách. O zavádění ovladačů do paměti se stará modul Drivers.dll.

Ovladač gmem.sys obsahuje funkce pro práci s pamětí. Zjišťuje stav paměti jádra a umožňuje ji číst a zapisovat do ní. Rozhraní k ovladači zapouzdřuje knihovna ssdt.dll. Název knihovny je odvozen od zkratky velmi důležité struktury jádra, kterou v této dokumentaci nazývám „Tabulka interních služeb systému.“ Ovladač je určen převážně k operacím s touto strukturou.

Hlavním cílem ovladače logptm.sys je sledování systému. Monitoruje vytváření a ukončování procesů a vláken a zavádění modulů do paměti. Udržuje ve své paměti seznam právě běžících procesů. Metodu vytváření tohoto seznamu stručně popíši v implementační části dokumentace. Věřím, že díky ní bude program schopen odhalit i procesy skryté moderními rootkity. Rozhraní k ovladači je uloženo v modulu logptm.dll.

## **Moduly pracující s okny**

---

<sup>2</sup> Z této výhody těží rozhraní Windows API a celý operační systém

<sup>3</sup> Nedostatečně dokumentovány jsou hlavně systémy Windows 9x.

Práci s okny obstarávají moduly OknaClient.dll a OknaServer.dll. Úzká spolupráce těchto knihoven umožňuje programu zjistit, která okna patří určitému procesu. Metoda získávání těchto údajů je docela neobvyklá a bude popsána v implementační části dokumentace.

## **Moduly pracující s procesy, vlákny a službami**

Do této kategorie patří PProc.dll, TProc.dll a SProc.dll. PProc.dll obsahuje sadu rozmanitých funkcí pro práci s procesy. Aplikace přes tento modulu zjišťuje seznam běžících procesů, stav paměti a dalších atributů. Zahrnutý jsou i funkce pro ukončení a pozastavení procesu. TProc.dll je nadstavbou nad operacemi s vlákny a funguje jak na Windows NT, tak i na Windows 98. Zahrnuje funkce pro pozastavení a ukončení vlákna a zjištění mnoha dalších informací. V modulu SProc.dll se nachází několik funkcí pro práci se službami.

K výše popsaným modulům zařadím DLL knihovnu ModuleOperation.dll. Prostředky skrývající se v tomto modulu lze odhadnout již z jeho názvu – jde převážně o funkce pracující s DLL knihovnami. Patří sem však i funkce pro vložení DLL knihovny do běžícího procesu a také funkce pro ověření, zda není kód knihovny pozměněn nějakým zlým programem.

## **Ostatní moduly nejnižší vrstvy**

Funkce a procedury zapouzdřující chybová hlášení a další všeobecnosti byly zařazeny do knihovny Procedures.dll.

Jednotka OSVersion obsahuje funkce pro určení verze Windows. Určení verze systému je velmi důležité, protože se od toho odvíjí, jaké prostředky Windows API je možno použít.

Jednotky WinNTFunctions a Win95Functions obsahují převážně nedokumentované funkce Windows API, které program využívá.

DisAsm.dll poskytuje rozhraní k jednoduchému disassembleru, který ocení zvláště programátoři.

V modulu Handles.dll se nacházejí podprogramy pro práci s právě používanými objekty jádra (otevřené soubory, sdílená paměť, používané klíče Registru, synchronizační objekty...). Modul FM.dll na tyto funkce navazuje – umožňuje získat obsah některých objektů jádra (souborů a sdílené paměti).

Jednotka Sorting obsahuje třídu TSortList, z níž jsou odvozeny téměř všechny třídy nacházející se v modulech střední vrstvy. TSortList zapouzdřuje funkce pro práci se seznamem položek. Umožňuje položky řadit podle více kritérií a zobrazovat je na formuláři. Jako řadící algoritmus bude použit Quicksort.

## **Moduly střední vrstvy**

Střední vrstvu tvoří jednotky, v nichž jsou deklarovány třídy, které obstarávají vizualizaci získaných informací (například zobrazení seznamu běžících procesů) a implementují funkce poskytované moduly nejnižší vrstvy. Jedinou výjimkou je knihovna Ar.dll, v níž je uloženo samotné jádro detekce rootkitů. Modul obsahuje funkce pro zjištění skrytých procesů, pro verifikaci Tabulky interních služeb systému a pro ověření, zda nejsou některé důležité DLL knihovny a ovladače napadeny škodlivým softwarem. O vizualizaci výsledků antirookitové kontroly se stará jednotka AntiRootkit a jednotka hlavního formuláře.

Protože všechny třídy určené k zobrazení nasbíraných informací mají velmi podobné úkoly, bude každá z nich krátce charakterizována v tabulce.

<b>Modul</b>	<b>Používán formulářem</b>	<b>Popis</b>
ClassDrivery	Hlavní formulář	Zjišťuje seznam zavedených ovladačů
ClassExporty	Informace o modulu	Zjišťuje funkce a symboly, které exportuje určitá DLL knihovna, nebo ovladač
ClassImporty	Informace o modulu	Zjišťuje údaje o funkcích, které určitá DLL knihovna nebo ovladač importuje
ClassKernelObjects	Informace o procesu	Zjišťuje, které objekty jádra jsou právě využívány určitým procesem, nebo v rámci celého operačního systému
ClassKernelPages	Hlavní formulář	Zobrazuje dostupnou paměť jádra a umožňuje číst její obsah
ClassMemPages	Informace o procesu	Zjišťuje stav paměti určitého procesu a umožňuje číst její obsah
ClassModule	Informace o procesu	Zapouzdřuje práci se seznamem modulů používaných určitým procesem
ClassModuleMemory	Informace o modulu	Ukazuje oblast paměti, kde se nachází DLL knihovna nebo ovladač
ClassModuleStrings	Informace o modulu	Zobrazuje seznam řetězců nacházejících se v určitém PE souboru.
ClassOkna	Informace o procesu	Zjišťuje informace o oknech patřících určitému procesu
ClassProcesy	Hlavní formulář	Pracuje se seznamem právě běžících procesů
ClassService	Hlavní formulář	Zobrazuje seznam nainstalovaných služeb
ClassSSDT	Hlavní formulář	Zobrazuje obsah Tabulky interních služeb systému (Systém Service Descriptor Table) a umožňuje ji opravovat.
ClassTokeny	Informace o procesu, Informace o vlákne	Zjišťuje oprávnění procesů a vláken
ClassVlakna	Informace o procesu, Informace o vlákne	Zjišťuje informace o vláknech běžících v kontextu určitého procesu.
PTMLogging	Informace o procesu	Zpracovává zprávy z knihovny logptm.dll (která

		je dostává z ovladače logptm.sys)
--	--	--------------------------------------

*Tabulka 3: Přehled modulů střední vrstvy*

## **Moduly svrchní vrstvy**

Jedná se o jednotky s uživatelským rozhraním a hlavní soubor celé aplikace. O uživatelském rozhraní již bylo řečeno vše podstatné v předchozí kapitole, takže stručně popíši, jaký kód se nachází v hlavním souboru.

Hlavní soubor programu příliš mnoho akcí neprovádí. Po spuštění aplikace provede nutnou inicializaci (nainstalování ovladačů, načtení DLL knihoven) a předá řízení do modulů s uživatelským rozhraním. Při ukončení procesu odinstaluje ovladače. Pokud je však program ukončen nestandardním způsobem (například Správcem úloh), ovladače nejsou odinstalovány a může dojít ke zpomalení systému. Konstrukce ovladačů však zajišťuje, že vše bude fungovat normálně, i když k takové události dojde.

## Implementace

### **Požadavky na operační systém a vývojové nástroje**

Process Inspector se skládá ze tří částí – z procesu běžícím v uživatelském režimu a dvou ovladačů. Celkové požadavky na operační systém jsou velmi striktní – Process Inspector funguje pouze na Microsoft Windows XP s nainstalovaným druhým Service Packem. Hlavní díl „viny“ nesou ovladače, ve kterých je použito několik nedokumentovaných postupů. Proti rootkitům však lepší možnost než nedokumentované postupy neexistuje. Samotný proces by po lehkých úpravách mohl běžet i pod Windows 9x a staršími Windows NT, samozřejmě s omezenými funkcemi.

Nároky na vývojové nástroje jsou díky rozsáhlosti aplikace trochu vyšší. Vlastní proces (uživatelské rozhraní) je programován v Borland Delphi 7 Enterprise. Na kompilování ovladačů užívám nástroj Microsoft Driver Development Kit (dále jen DDK). Při psaní kódu ovladačů jsem si vystačil se základním textovým editorem, který ve Windows nalezneme – s Poznámkovým blokem.

Při testování Process Inspectoru proti rootkitům někdy využívám virtualizačních možností nástroje Microsoft Virtual PC 2004.

### **Řešení některých problémů**

V této části vysvětlím, jak jsem řešil některé obecné i konkrétní problémy, které se vyskytly během vývoje Process Inspectoru. Pochopení níže popsanych problémů vyžaduje určité znalosti operačního systému Windows NT a také znalosti v oboru vývoje ovladačů. Vše potřebné je popsáno v přílohách k této dokumentaci.

### **Komunikace uživatelského rozhraní s ovladači**

Komunikace s ovladači se při vývoji ukázala jako klíčový problém. Naštěstí rozhraní Windows API mě opět nezklamalo. Pokud uživatelské rozhraní potřebuje poslat zprávu ovladači, učiní tak pomocí funkce DeviceIoControl. Tato funkce umožňuje poslat nejen předem domluvený kód zprávy, ale také připravit vstupní a výstupní buffer, který bude ovladači k dispozici. Ovladač se podle kódu zprávy rozhodne, jakou činnost má provést. Pokud k této činnosti jsou potřeba vstupní data, jsou načtena ze vstupního bufferu, který byl připraven uživatelským rozhraním aplikace. Po dokončení úkolu ovladač naplní aplikací připravený výstupní buffer a informuje ji o úspěchu nebo neúspěchu.

Funkce DeviceIoControl umožňuje několik typů komunikace mezi procesem a ovladačem. Detaily jednotlivých typů komunikace se zde nebudu zabývat, zájemci si mohou všechny podrobnosti přečíst v DDK.

Process Inspector využívá tzv. „bufferované metody“ komunikace. Tento typ komunikace se používá pro přenášení menších objemů dat, protože není příliš šetrný k nestránkované paměti počítače. Průběh komunikace „bufferovanou metodou“ lze rozdělit do těchto kroků:

- Aplikace alokuje vstupní a výstupní buffer. Do vstupního bufferu zapíše potřebná data a zavolá DeviceIoControl s patřičným kódem zprávy pro ovladač.
- Správce vstupně-výstupních operací alokuje oblast nestránkované paměti. Velikost oblasti je rovna většímu z bufferů připravených aplikací. Do alokované oblasti Správce přepokopí obsah vstupního bufferu a předá řízení ovladači.
- Ovladač načte vstupní data z oblasti alokované Správcem vstupně-výstupních operací. Po dokončení práce jsou výstupní data zapsána do stejného paměťového místa. Vstupní data jsou tedy přepsána výstupními. Jelikož je tato oblast z nestránkované paměti, ovladač může tuto paměť měnit za jakýchkoli okolností. Nakonec ovladač nastaví návratový kód a určí velikost výstupních dat (nemusí být shodná s velikostí výstupního bufferu, ale nesmí být větší).
- Správce vstupně-výstupních operací zkopíruje výstupní data z nestránkované paměti do aplikací připraveného bufferu, uvolní teď již zbytečnou nestránkovanou oblast a předá řízení aplikaci.
- Aplikace zkontroluje návratový kód. V případě úspěchu obsahuje výstupní buffer validní data, se kterými je možné dále pracovat.

Funkce DeviceIoControl poslouží výborně, jestliže je iniciátorem komunikace uživatelské rozhraní. Pokud však komunikaci zahajuje ovladač, nelze ji použít. Bohužel, taková situace při programování Process Inspectoru skutečně nastala. Pro řešení tohoto problému jsem se rozhodl využít sdílenou paměť a synchronizační objekty zvané událost (anglicky „event“).

Objekt událost si může představit jako pomyslnou šňůru mezi dvěma (nebo více) programy. Jeden z programů vykonává svůj kód, ostatní čekají na signál. Když program dokončí určitou akci (například naplní sdílenou paměť výsledky své práce), zatáhne za pomyslný provaz a tím probudí ostatní programy k činnosti. Tímto postupem lze například zajistit, aby program, kterému potřebujeme předat data ve sdílené paměti, nenačetl tyto data předčasně (když se ještě ve sdílené paměti nenacházejí, nebo jsou jen zcela zapsána). A přesně tento princip je využit při komunikaci ovladače s Process Inspektorem.

Nyní známe dostatek teorie, abychom pochopili body, kterými je realizována komunikace ovladače s uživatelským rozhraním:

- Kód ovladače zapíše obsah zprávy do sdílené paměti (sdílená paměť a události byly vytvořeny při inicializaci ovladače) a pomocí události A dá signál uživatelskému rozhraní. Pak čeká, dokud zpráva není zpracována.
- Uživatelské rozhraní obdrží signál, že sdílená paměť obsahuje zprávu. Načte obsah sdílené paměti a zpracuje ji. Potom je pomocí události B poslán signál ovladači, že zpráva byla přijata.
- Ovladač zachytí signál události B. Nyní může bezpečně zapsat novou zprávu do sdílené paměti a opět ji odeslat pomocí události A.

Když je zapisována zpráva do sdílené paměti, zapisující vlákno využívá další události k zajištění, aby žádné jiné vlákno nezapisovalo jinou zprávu ve stejný čas. To by vedlo k poškození obsahu zprávy.

## Přenos dat v rámci ovladače a problém priorit

Aby čtenář porozuměl tomuto problému, musí mít určité znalosti v oboru vývoje ovladačů. Všechny potřebné znalosti můžete načerpat z přílohy A.

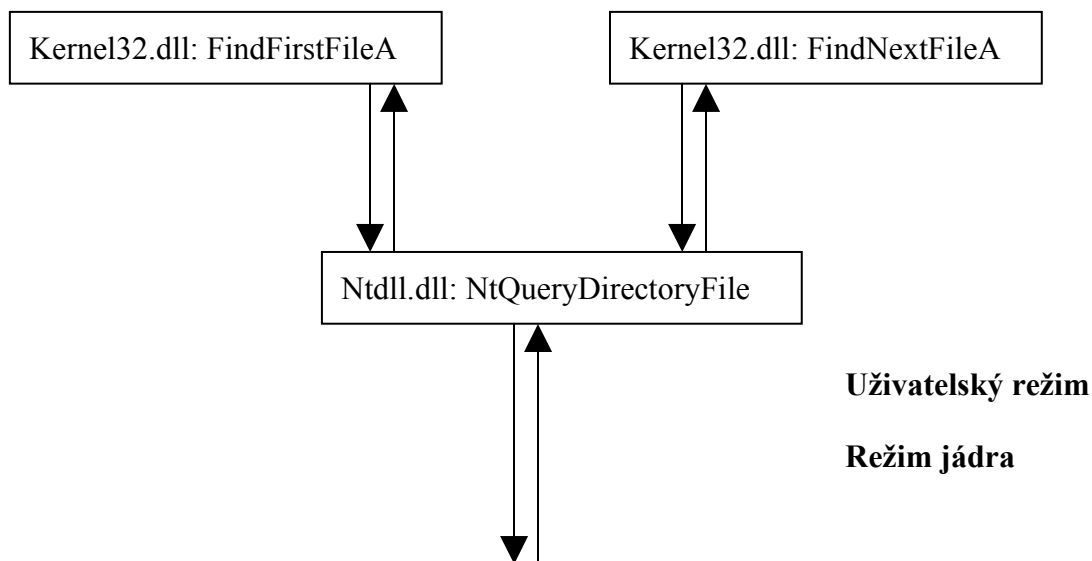
V minulé kapitole jsme se zabývali problémem komunikace ovladače s uživatelským rozhraním. Může však nastat situace, kdy ovladač právě nasbíraná data nemůže pomocí výše popsaného postupu aplikaci předat. Tato situace nastane, když kód sbírající data běží s prioritou DISPATCH\_LEVEL nebo vyšší. Konkrétně se zde budeme zabývat prioritou DISPATCH\_LEVEL, vyšší priority přináší ještě další komplikace. Zatím se žádné problémy s vyššími prioritami řešit nemusely.

Jak již víte, Process Inspector také obsahuje detektor rootkitů. Podstatnou částí detektoru je vyhledávání skrytých procesů. Aby bylo vyhledávání co nejučinnější, je přeměrováno jedno z důležitých softwarových přerušení na jednu z rutin ovladače logptm.sys. Tato rutina sbírá některé velmi důležité informace. Nemůže je však ihned předat ke zpracování, protože běží s prioritou DISPATCH\_LEVEL. Při tak vysoké prioritě již nelze používat standardních synchronizačních metod. Pokud kód běží v DISPATCH\_LEVEL, nemůže čekat na to, až bude zatáhnuto za pomyslný provaz nějaké události (event), protože kód s takovou prioritou lze přerušit jen hardwarovým přerušením. Krom toho, jakýkoliv kód s prioritou DISPATCH\_LEVEL by měl být vykonán co nejrychleji, aby nebránil procesoru v multitaskingu. Při DISPATCH\_LEVEL také není možné pracovat se sdílenou pamětí, protože ta je stránkována.

Problém byl vyřešen následujícím způsobem: rutina ovladače logptm.sys, která je vykonávána s prioritou DISPATCH\_LEVEL, pouze alokuje nestránkovanou paměť a do ní uloží potřebná data. Právě alokovanou oblast připojí do obousměrného spojového seznamu a skončí. Obousměrný seznam je zpracováván speciálním vláknem (které má prioritu PASSIVE\_LEVEL) a data jsou odesílána přes sdílenou paměť uživatelskému rozhraní. Je pravda, že řečené vlákno nestačí zprávy odesílat dostatečně rychle, takže spojový seznam narůstá. Experimentálně však bylo zjištěno, že to na běhu systému nemá žádné negativní následky.

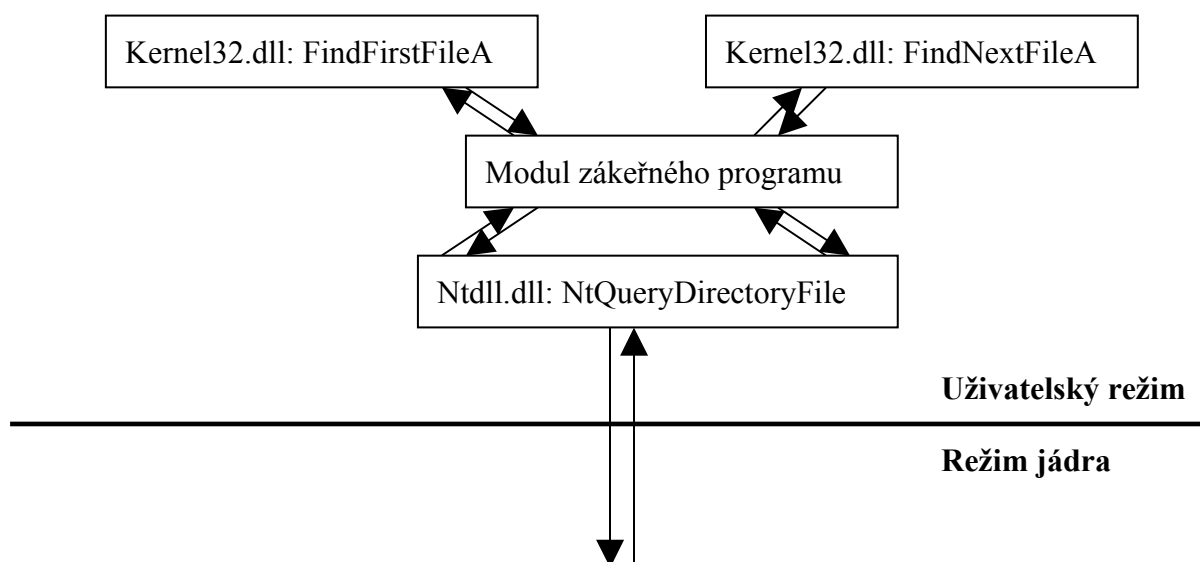
## Detekce přesměrování toku programu

Nejdříve si ujasněme, co to vlastně přesměrování toku programu je. Představme si aplikaci typu souborový manažer. Tato aplikace se právě snaží načíst obsah adresáře „C:\Documents and Settings\“. K tomuto účelu použije standardní dvojici funkcí Windows API FindFirstFileA a FindNextFileA. Obě funkce jsou exportovány knihovnou kernel32.dll.



**Obrázek 3:** Průběh funkcí *FindFirstFileA* a *FindNextFileA*

Na obrázku vidíme průběh funkcí *FindFirstFileA* a *FindNextFileA*. Obě po vykonání určitých operací zavolají funkci *NtQueryDirectoryFile* z knihovny *ntdll.dll*. *NtQueryDirectoryFile* zavolá jádro systému. Jádro zjistí seznam souborů v *C:\Documents and Settings\* a výstup opět předá funkci *NtQueryDirectoryFile*. Výstup je opačnou cestou poslán souborovému manažeru. Obrázek však může vypadat i jinak.



**Obrázek 4:** Průběh volání *FindFirstFileA* a *FindNextFileA* podruhé

Rozdíl mezi oběma obrázky je patrný na první pohled. Na druhém obrázku vidíme, že v rámci volání výše zmiňovaných funkcí se vykonává i kód naprosto cizí aplikace. Kód této aplikace může změnit data vrácená podprogramem NtQueryDirectoryFile a tím například skrýt některé soubory před souborovým manažerem. Aplikace *přesměrovala tok programu*. V odborném jazyce se říká, že *umístila hák* na funkce FindFirstFileA a FindNextFileA, nebo že tyto funkce byly *zahákovány*. Více o hákování se dozvíte v příloze věnované rootkitům.

Jak ale zjistit, že určitá funkce byla zahákována? Některé metody hákování jsou založené na přepsání prvních bytů hákovaného podprogramu. Tyto změny v kódu Process Inspector detekuje. Algoritmus je velmi prostý – program prozkoumá začátky všech funkcí exportovaných nejdůležitějšími knihovnami systému (kernel32.dll, user32.dll). Podobný úkon pak provede i v jádře – prozkoumá však soubory všech ovladačů.

## Detekce skrytých procesů

Detekce skrytě běžících procesů patří mezi dovednosti každého detektoru rootkitů, Process Inspector nevyjímaje. Způsob detekce není nijak složitý – program zjistí seznam běžících procesů několika metodami. Porovnáním výsledků získá procesy, jež běží skrytě. Mezi používané metody patří:

- Použití funkcí Windows API CreateToolhelp32Snapshot, Process32First, Process32Next.
- Volání funkce EnumProcesses z knihovny PSAPI.DLL.
- Volání funkce NtQuerySystemInformation z knihovny ntdll.dll.
- Prozkoumání právě používaných objektů jádra (např. otevřených souborů).
- Zkoumání obousměrného spojového seznamu struktur EPROCESS (více v příloze o rootkitech).
- Hákování instrukce SYSENTER (de facto hákování přerušení) (více opět v příloze B).

V nových verzích programu budou implementovány ještě další, pokročilejší metody detekce.

## Získání seznamu oken cizího procesu

K zjištění, jaká okna patří určitému procesu, se běžně používá funkce Windows API EnumWindows. Má však své nedostatky – pokud například cílový proces běží pod jiným uživatelem než proces funkci volající, výsledky volání jsou nepravdivé – rutina nenajde nic. To však není jediný problém; některé informace o okně může zjistit jen proces, který cílové okno vlastní. Z těchto důvodů jsem se rozhodl pro poněkud netradiční techniku, které se většinou používá ve virech a jim podobných programech. Vše je zapouzdřeno v modulech OknaClient.dll a OknaServer.dll. Postup získání seznamu oken se skládá z následujících kroků:

- Modul OknaClient.dll vytvoří dva synchronizační objekty typu událost (event) a jednu oblast sdílené paměti.

- Modul OknaClient.dll zajistí, aby cílový proces získal přístup k objektům vytvořeným o v předchozím kroku.
- Modul OknaServer.dll je vložen do cílového procesu. Při své inicializaci si otevře přístup k dvěma událostem (event) a ke sdílené paměti. OknaServer.dll se stane de facto serverem, čekajícím na požadavky klienta.
- OknaClient.dll naplní sdílenou paměť požadavkem na zjištění všech oken cílového procesu. Dokončení práce signalizuje pomocí události A.
- OknaServer.dll zachytí signál, přečte si obsah sdílené paměti a vyplní jej seznamem oken cílového procesu. K zjištění tohoto seznamu může bezpečně použít funkci EnumWindows, protože běží v kontextu cílového procesu. Dokončení úkolu signalizuje pomocí události B.
- OknaClient.dll přečte ze sdílené paměti výsledky. Celý proces se nyní může opakovat. OknaClient.dll může server žádat o informace k jednotlivým oknům cílového procesu.

Oba moduly spolu komunikují naprosto stejným způsobem jako ovladač logptm.sys s uživatelským rozhraním.

## Testování

Testování je velmi důležitá etapa vývojového cyklu aplikace, u systémových programů to platí dvojnásob. Program musí být důkladně otestován, aby bylo možno říci, že bude správně fungovat na všech počítačích, které vyhovují požadavkům, jenž byly popsány v implementační části dokumentace. Chybně fungující aplikace by mohla způsobovat pády operačního systému. Proto se snažím program testovat na několika počítačích zároveň.

### *Testy jednotlivých funkcí*

Většina funkcí dostupných v Process Inspectoru pracuje s procesy, DLL knihovnami, službami, ovladači a jinými objekty, jejichž poškození může ovlivnit chod operačního systému. Proto je testování ovlivněno následujícími podmínkami:

- Funkce pracující s procesy jsou testovány na denně používaných aplikacích. Nejčastějším „obětním beránkem“ je známý souborový manažer Total Commander. Na systémových procesech se provádí testy velmi zřídka.
- Funkce pracující s ovladači jsou testovány buď na živém systému, nebo v rámci virtuálního počítače. Pokud je během testů odhalena chyba, snažím se ji co nejrychleji lokalizovat a odstranit; časté pády operačnímu systému příliš neprospívají.
- Funkce pracující se službami budou testovány na vlastnoručně vytvořených testovacích službách.
- Funkce pracující s tabulkou interních služeb systému jsou testovány na živém systému. Aby byly testy co nejprůkaznější, používám programy, které s touto důležitou strukturou manipulují.

### *Křest ohněm*

Aby byly důkladně prověřeny antirootkitové schopnosti programu, rozhodl jsem se postavit jej proti testovacím rootkitům. Testovací rootkity mají tu výhodu, že nepoškozují váš systém, jsou určeny pouze pro testovací účely. Rootkity používané ve skutečném malware používají většinou techniky obsažené v testovacích rootkitech. Program bude zkoušen s rootkity, jejichž přítomnost může principiálně odhalit. Budou provedeny testy s těmito rootkity:

- **Vanquish.** Rootkit běžící pouze v uživatelském režimu. Skrývá soubory obsahující řetězec „vanquish“ ve svém názvu.
- **AFX Rootkit 2005.** Opět rootkit operující jen v uživatelském režimu. Umožňuje skrývat, procesy, soubory, klíče registru... prostě téměř cokoliv. K dosažení svého cíle používá technik hákování<sup>4</sup>. Jedná se snad o jediný testovací rootkit naprogramovaný v Delphi.

---

<sup>4</sup> Hákování a další techniky používané autory rootkitů a malwaru jsou vysvětleny v příloze B

- **NTRootkit.** Operuje v režimu jádra. Manipuluje s tabulkou interních služeb systému a dosahuje tím například skrývání procesů. Jedná se vůbec o první veřejný testovací rootkit.
- **FU.** Rootkit-ovladač. Skrývá procesy a ovladače technikou DKOM. Vybraným procesům dokáže zvyšovat privilegia.
- **FuTo.** Pokračování rootkitu FU. Jeho schopnosti jsou totožné s předchůdcem.
- **Phide\_ex.** Skrývá proces technikou DKOM. Velmi moderní a pokročilý rootkit. Pro Process Inspector bude opravdovou výzvou.
- **Rustock.** Rustock nepatří mezi testovací rootkity. Jedná se o ovladač pe386.sys, který bývá obsažen ve skutečném malwaru. Dodnes jsou známy dvě varianty tohoto rootkitu – Rustock A a Rustock B. Je možné, že brzy spatří světlo světa i třetí varianta – Rustock C. Proslýchá se, že tuto novou variantu nebude možné dnešními prostředky detekovat, je-li přítomna v běžícím operačním systému. Jediná možnost spočívá v zkontrolování infikovaného harddisku offline. Testy s dosud známými variantami budou prováděny za velmi přísných bezpečnostních opatření.

## Výsledky

V následující tabulce uvádím, jak si program vedl proti některým rootkitům.

Rootkit	Výsledek
AFX Rootkit 2006	Program zjistil změny v systémových knihovnách způsobené hákováním některých důležitých rutin. Změny nelze opravit
Vanquish	Odhaleny změny systémových knihoven (háky důležitých rutin). Změny nelze opravit
NTRootkit	Odhaleny změny SSDT. Program dokáže SSDT opravit a tím rootkit zneškodnit
Fu	Odhalen skrytý proces a podařilo se o něm zjistit i některé podrobnosti. Proces bylo možné násilně ukončit.
FuTo	Odhalen skrytý proces, ale nepodařilo se o něm zjistit nic kromě PID. Proces nelze ukončit ani s ním jinak manipulovat
Phide_ex	Testování se nezdařilo. Rootkit je na virtuální počítači velmi nestabilní a jeho běh končí modrou obrazovkou. Pokud se tak náhodou nestane, k pádu systému dojde po spuštění Process Inspectoru

*Tabulka 4: Výsledky testů*

## Závěr a diskuze

### Závěr

Při vývoji Process Inspectoru se moje programátorské znalosti a schopnosti dostaly na mnohem vyšší úroveň, než tomu bylo předtím. Naučil jsem se vyvíjet ovladače a řešit některé problémy s tím spojené. Dozvěděl jsem se mnoho informací nejen o jádru systému Windows NT. Při programování jsem poprvé použil některé netradiční, avšak velmi efektivní metody.

Při vývoji této aplikace jsem si plně uvědomil, jak důležité je plánovat dopředu. Ač se to zpočátku zdálo nepředstavitelné, plánování ušetřilo mnoho času a díky tomu jsem stihl do programu zabudovat docela slušný detektor rootkitů. Do budoucna však vím, že plánování je třeba uplatňovat ještě mnohem více.

Možných vylepšení se nabízí celá řada. Program nevyhledává skryté soubory, ovladače ani klíče registru. Nedokáže opravit poškozené systémové knihovny. Nedokáže monitorovat start systému. Rovněž chybí možnost monitorovat souborový systém. Vyhledávání skrytých procesů by také mělo být ještě vylepšeno. Program nedokáže zabránit spuštění uživatelem definovaných procesů. Chybí toho ještě mnoho, ale to se časem změní.

### Diskuze

V této části porovnáím funkce a možnosti programu s aplikacemi, jenž slouží k podobnému účelu jako Process Inspector a které jsem zevrubně popsal v úvodní části dokumentace. Program mezi nijak zvlášť nevyniká, avšak také nezůstává pozadu. Proberme si tedy jednotlivé skupiny funkcí a porovnávejme.

### Vyhledávání skrytých procesů

BlackLight, IceSword i RootkitUnhooker dokáží detekovat skryté procesy. Rozdíl se projeví při hledání procesů skrytých rootkitem FuTo. Process Inspector dokáže zjistit jen PID skrytého procesu, stejných výsledků dosahuje BlackLight. IceSword a RootkitUnhooker naopak zobrazí jméno procesu a další informace, avšak PID zjistit nedokáží.

### Kontrola systémových knihoven

Tato funkce u programů IceSword a BlackLight zcela chybí, zbývá tedy jen RootkitUnhooker, který má ji implementovanu na mnohem vyšší úrovni než Process Inspector. Dokáže totiž odhalit i hákování tabulky importů. Oba jsou schopni detekovat „vkládané“ háky. RootkitUnhooker navíc dokáže tyto háky odstranit.

## Zobrazování obsahu paměti jádra

Žádný z programů popisovaných v úvodní části touto funkcí nedisponuje. RootkitUnhooker umožňuje získat obsah paměti, ve které se nachází ovladače, paměť neasociovanou s žádným ovladačem však nikoli. Toto je jedna z mála oblastí, kde je Process Inspector silnější, avšak je stále co vylepšovat.

## Kontrola tabulky interních služeb systému (SSDT)

S touto tabulkou umí pracovat jak IceSword, tak RootkitUnhooker. BlackLight se specializuje pouze na vyhledávání skrytých souborů a procesů. IceSword dokáže zaregistrovat jen přímou změnu této struktury. Pokud je však použito „vkládaných“ háků, je slepý. Je k tomu však poměrně dobrý důvod – IceSword „vkládaných“ háků využívá ke své ochraně. Kdyby je zobrazil, odhalil by svoje vlastní praktiky. RootkitUnhooker a Process Inspector odhalí i „vkládané“ háky, RKU je dokáže i odstranit.

## Práce s procesy

V této oblasti RootkitUnhooker, IceSword i BlackLight silně pokulhávají. Zde vládnu správcé úloh jako Process Explorer nebo Process Inspector. RootkitUnhooker nabízí pouze několik zajímavých funkcí (násilné ukončení, smazání hlavního modulu), IceSword je na tom trochu lépe – zobrazí i seznam DLL knihoven a vláken. Dokáže si poradit i se čtením a zápisem do paměti. Informace, které o procesech zjišťuje Process Explorer, jsou velmi podrobné a týkají se mnoha oblastí. Process Inspector za tímto programem příliš nezaostává – navíc obsahuje jednoduchý disassembler, pomocí kterého každý může nahlédnout do nedokumentovaných rutin jádra operačního systému. Údaje sbírané Process Explorerem se více hodí normálnímu uživateli (například využití CPU), Process Inspector je zaměřen spíše na pokročilé uživatele a programátory. Process Explorer zobrazuje seznam procesů do stromu, ze kterého je vidět, kdo je otcem toho či onoho programu. Process Explorer je také lepší v zobrazování informací o právě využívaných objektech jádra (otevřených souborech, klíčních registru atd.) – používá k tomu účelu speciální ovladač.

## Závěr diskuze

Process Inspector není nejlepším programem ani v oblasti správců úloh ani v oblasti detektorů rootkitů. Je směsicí obojího, ale má blíže ke správci úloh. I přesto obsahuje funkce, které nejsou obsaženy v jiných detektorech rootkitů a při vyhledávání skrytých procesů příliš nezaostává.

## Použitá literatura

HOGLUNG, G. and BUTLER, J. *Rootkits: Subverting The Windows Kernel*. 1st edition: Addison-Wesley, 2005. ISBN 0321294319

RUSSINOVICH, M. E. and SALOMON, D. A. *Microsoft® Windows® Internals*. 4th edition: Microsoft Press, 2004. ISBN 0735619174.

PIETREK, M. *Windows 95 Systém Programming Secrets*. 2nd edition. IDG Books Worldwide Inc., 1995. ISBN 1568843186.

NEBBET, G. *Windows NT / 2000 Native API Reference*. 1st edition. Sams, 2000. ISBN 1578701996

TEIXEIRA, S. a PACHECCO, X. *Borland Delphi: průvodce vývojáře, kniha IV*. 1. vydání. UNIS Publishing s. r. o, 1999. ISBN 8086097366.

MANNOVÁ, B. a VOSÁTKA, K. *Řízení softwarových projektů*. Nakladatelství ČVUT, 2005. ISBN 8001032973