Implementace systémů HIPS

Martin Dráb

martin.drab@secit.sk

Upozornění

Tento text vznikl jako příprava mojí přednášky na konferenci **SOOM.cz Hacking & Security Konference**. Vytvořil jsem jej podle slidů prezentace, které lze chápat jako osnovu. Text obsahuje přesnější informace než prezentace, která si kladla za cíl hlavně nezamotat hlavu posluchačů údaji, jenž sice jsou správné, ale pro přednášku zbytečně složité. Tím chci říci, že nejen jednotlivé slidy, ale i tento text, podávají některá témata přednášky značně zjednodušenou formou.

Text nepopisuje slidy věnované rozhraní určenému k ochraně registru. Hlavním důvodem této absence je fakt, že jsem o tomto rozhraní neplánoval říci více, než mám uvedeno v prezentaci.

Pokud vás zajímá, jak popisované techniky implementace systémů HIPS a mechanismy operačního systému Windows do detailu fungují, doporučuji nahlédnout do poslední kapitoly textu, kde najdete odkazy na zajímavé stránky, knihy a dokumenty.

Úvod

Tuto přednášku jsem se rozhodl věnovat problematice programování systémů HIPS na operačních systémech Microsoft Windows a případně dalším konceptům, které sice nelze využít přímo ke kontrole činnosti aplikací a systému, ale i tak přinášejí některé zajímavé možnosti.

Vyjasnění cílů

Abychom všichni mysleli pod pojmem „systém HIPS“ to samé, rozhodl jsem se jej na začátku přednášky zadefinovat. Budu jím označovat aplikaci, která si klade za cíl monitorovat dění v operačním systému a chování jednotlivých spuštěných programů, hlásit podezřelé aktivity a případně je blokovat. Takové aplikace se postupně staly běžnou součástí bezpečnostních balíků.

Pravda sice je, že Microsoft Windows, stejně jako další velké operační systémy, disponuje bezpečnostním modelem, který dovoluje jednotlivým uživatelům definovat oprávnění k různým objektům. Mnoho domácích uživatelů však stále pracuje pod administrátorským účtem, takže bezpečnostní model není příliš efektivní. S příchodem Windows Vista se sice objevila náprava této situace v podobě mechanismu Kontroly uživatelských účtů (User Account Control – UAC), řada uživatelů jej však vypíná s tím „aby je neotravoval.“ Systémy HIPS tedy z jistého úhlu pohledu nahrazují funkce bezpečnostního modelu.

Zajímavá otázka zní, jakým způsobem se systémy HIPS implementují. Odpověď na ni se pokusím poskytnout v této přednášce. Budu se věnovat hlavně technikám, které se používaly převážně v minulosti a dnes od nich antivirové firmy ustupují. To ale neznamená, že tyto techniky nemají stále své místo a že nemohou začít znovuzrození. Důvodem ústupu je hlavně nemožnost použití těchto technik na 64bitových verzích Windows a také dostupnost nových rozhraní, která umožňují dosáhnout podobných výsledků pohodlnější a legitimní (dokumentovanou) cestou. O některých rozhraních si také povíme něco podrobnějšího. Pokusím se také vysvětlit některé základní principy, na kterých Windows stojí, aby byla přednáška pochopitelná i pro ty, co se v těchto vodách neorientují.

Na jaké techniky se tedy zaměříme:

- Modifikace kódu
- Modifikace tabulek systémových volání
- Direct Kernel Object Hooking (DKOH)
- OB Filtering model

Nejprve je ale potřeba vysvětlit si několik věcí o tom, jak operační systémy fungují.

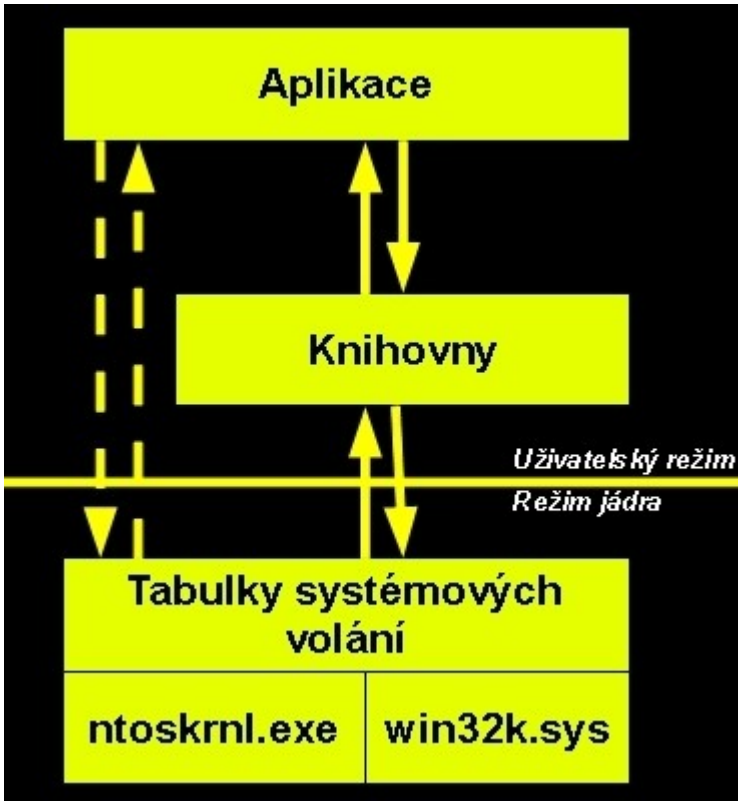
Základní informace

Na obrázku 1 vidíte poměrně zjednodušené schéma architektury operačního systému. Z obrázku je patrné, že aplikace k plnění svých úkolů téměř výhradně využívají služeb knihoven, které si můžete představit jako souhrny rutin zodpovědných za provedení různých operací. Knihovny za účelem vyřízení některých požadavků od aplikací využívají služeb jádra operačního systému. Požadavky mu předávají prostřednictvím mechanismu *systémových* volání, o který si podrobněji popíšeme později. Takováto struktura je dána tím, že jak kód aplikací, tak kód knihoven běží v *uživatelském režimu* procesory, který zakazuje provádění některých instrukcí (komunikace s hardware, práce s virtuální pamětí, změna stavu procesoru).

Pokud knihovna potřebuje provést operaci, kterou ji uživatelský režim procesoru zakazuje, předá požadavek jádru operačního systému, které je v našem obrázku reprezentováno pouze dvěma ovladači: *ntoskrnl.exe* (procesy, vlákna, registry, soubory...) a *win32k.sys* (grafické uživatelské

rozhraní). Tabulky systémových volání pro nás zatím představují černou skříňku, která zaručuje doručování požadavků správným ovladačům.

Všimněte si také, že aplikace nemusí s jádrem operačního systému komunikovat prostřednictvím knihoven, ale mohou tak činit i přímo, ač je tento postup nedokumentovaný a jen málokdy používaný, zvláště u legitimních aplikací.



Obrázek 1: Architektura operačního systému

Modifikace kódu

Nyní konečně nastal čas si povědět o první technice, která se při implementaci systémů HIPS v minulosti hojně používala a stále nachází uplatnění i v dnešní době. Jedná se o modifikace kódu. Tato technika funguje tak, že součást systému HIPS pozmění kód rutin zodpovědných za provádění operací, nad kterými chce HIPS převzít kontrolu. Obvykle je provedená změna taková, že místo původní rutiny je vykonán kód podprogramu součásti systému HIPS, který rozhodne, zda příslušnou operaci povolit či blokovat (může si přitom vyžádat i reakci od uživatele). Pokud operace dostane zelenou, systém HIPS předá řízení kódu původní rutiny, čímž dochází k faktickému provedení dané operace. V některých případech následuje i úprava výsledků dané operace, ačkoliv k tomuto scénáři dochází obvykle u malware (skrývání souborů, procesů...).

Velkou výhodou výše popsané techniky je, že ji lze provozovat prakticky na libovolné úrovni; v knihovnách, v ovladačích jádra, v samotné aplikaci. Podle místa se odvíjí účinnost. Pokud například za účelem zabránění násilného ukončení vaší aplikaci modifikujete kód příslušných knihovních rutin, vaše ochrana není stoprocentně efektivní, protože ji aplikace obejde tím, že bude komunikovat přímo s jádrem operačního systému, což je proveditelné, jak vidíte na obrázku 1.

Další výhodu představuje fakt, že modifikace kódu můžete využít pro implementaci ochrany prakticky ve všech oblastech (procesy, souborový systém, registry...). Záleží pouze na tom, jaké rutiny modifikujete a kde.

Modifikace kódu s sebou ale přináší také spoustu problémů. Jmenujme například závislost na

architektuře procesoru. Jelikož se jedná de facto o přepisování instrukcí, musí systém HIPS obsahovat speciální modul pro každou podporovanou architekturu (u Windows jde zatím o x86 a x64, popř. IA64).

Omezení použití modifikace kódu se dostavilo na 64bitových platformách kvůli technologii Patchguard, která kontroluje integritu některých modulů jádra (například *ntoskrnl.exe*).

Další problém této techniky spočívá v tom, že modifikací kódu můžete (nevědomě) drobně pozměnit její chování, což může zavinit nekompatibilitu některých aplikací. Jako příklad si uveďme AntiWPA, celkem známý program, který zajišťoval bezproblémový běh neaktivovaných kopií Windows XP a Windows Server 2003 i po vypršení zkušební doby.

AntiWPA

Program modifikoval rutinu *NtQuerySystemInformation* v knihovně ntdll.dll v procesu winlogon.exe. Tato rutina dovoluje aplikacím dozvědět se mnoho informací o systému, na kterém se nacházejí, včetně toho, zda je spuštěn v nouzovém režimu či nikoli. To je důležité, protože *winlogon.exe* v nouzovém režimu vypršení zkušební doby nekontroloval.

AntiWPA upravil rutinu *NtQuerySystemInformation* tak, že na dotaz ohledně nouzového režimu vždy odpoví, že tento režim je aktivní. Jelikož kopie knihovny v ostatních procesech nejsou modifikovány, touto iluzí trpí pouze winlogon.exe, takže většina operačního systému nebyla nijak negativně ovlivněna.

Rutina byla naneštěstí modifikována tak, že při ostatních dotazech (kdy se volající neptal na nouzový režim) došlo k nekonečné smyčce, což znamenalo zatuhnutí příslušného vlákna v procesu winlogon.exe. Na čistých instalacích Windows toto přídavné chování nevadilo, protože winlogon.exe tuto rutinu používal pouze na zjištění, zda systém neběží v nouzovém režimu. Po instalaci bezpečnostního balíku Norton Internet Security však byla situace zcela jiná, došlo k zatuhnutí a uživatel nikdy neviděl svoji pracovní plochu.

Modifikace tabulek systémových volání

Již jsem se zmínil o tom, že kód aplikací je prováděn v uživatelském režimu procesoru, který jim zakazuje provádět některý instrukce. Z tohoto důvodu aplikace (ať už přímo nebo prostřednictvím knihoven) prostřednictvím systémového volání požádá jádro, zda nemůže příslušnou privilegovanou operaci vykonat. Jádro požadavku může vyhovět, ale také nemusí (například kvůli bezpečnostnímu modelu). Kód ovladačů jádra je vykonáván v *privilegovaném režimu* procesoru (režimu jádra), který neklade žádná omezení co se týče povolených instrukcí.

Systémová volání a systémy HIPS

Standardní průběh systémového volání vypadá tak, že knihovny přeloží požadavek aplikace tak, aby mu jádro rozumělo, a zajistí vykonání speciální instrukce (SYSENTER, SYSCALL, INT 0x2E). Tato instrukce způsobí přechod procesoru do privilegovaného režimu a předá řízení kódu pro zpracování systémových volání, který byl operačním systémem určen během startu. Úkolem tohoto kódu je zjistit, jakou operaci potřebuje aplikace provést, a předat řízení příslušné rutině.

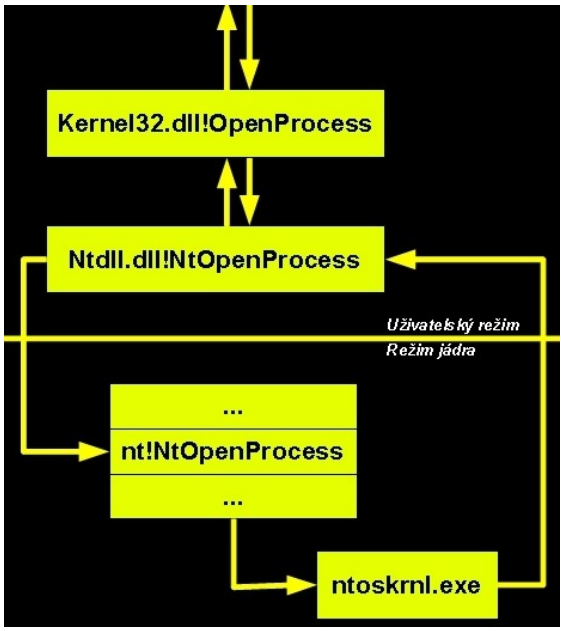
Informace o typu operace je z uživatelského režimu do privilegovaného režimu obvykle přenášena v podobě číselného identifikátoru v některém z registrů procesoru. Jádro uchovává adresy rutin zodpovědných za jednotlivé operace v datové struktuře zvané *tabulka systémových volání*. Můžete si ji představit jako černou skříňku, která pro zadané číslo vyplivne adresu rutiny zodpovědné za vykonání příslušné operace. Na 32bitových verzích Windows je implementována jako pole adres rutin a číslo operace se používá jako index. Na 64bitových platformách je implementace o něco složitější.

Vybraná rutina nejprve musí ověřit platnost argumentů operace a překopírovat jejich obsah do paměti přístupné pouze z privilegovaného režimu (paměti jádra). Tento krok je nutný, protože při přechodu do režimu jádra nedochází ke kopírování paměti s argumenty pro operaci, ale potřebná data (obvykle ve formě odkazů/adres) jsou předávána pouze v registrech procesoru, případně překopírována před zavoláním samotné rutiny (kopírují se však jen odkazy či přímé hodnoty, nikoli obsah paměti, na kterou dané odkazy vedou). Zbytek se nachází v části paměti přístupné jak z privilegovaného režimu, tak z uživatelského režimu (jinak by je tam aplikace ani nemohla zapsat). Pod pojmem *ověřování platnosti a kopírování do paměti jádra* myslíme proces postupného kopírování všech argumentů operace na místo, kam aplikace nemá přístup. Aplikace totiž během samotného systémového volání může obsah argumentů měnit, případně uvolnit paměť, ve které se nacházejí, s úmyslem způsobit systému HIPS co největší problémy.

Výsledky operace jsou vráceny aplikaci do uživatelského režimu. Pro návrat se použije opět speciální instrukce (SYSEXIT, SYSRET, IRET).

Na obrázku 2 vidíte průběh systémového volání pro aplikaci, která se pokouší získat přístup k nějakému procesu. Za tímto účelem volá rutinu *OpenProcess* z knihovny *kernel32.dll*. Tato rutina započne překlad požadavku aplikace na systémové volání. Předá řízení funkci *NtOpenProcess* z knihovny *ntdll.dll*, jenž překlad požadavku dokončí. Do registrů procesoru nastaví číselný identifikátor operace a odkaz na argumenty a provede speciální instrukci pro přechod do privilegovaného režimu.

Jádro operačního systému na základě číselného typu operace vyhledá v tabulce systémových volání příslušnou rutinu. Na čistém systému se tato rutina jmenuje též *NtOpenProcess*. Rutina ověří platnost argumentů a překopíruje je do paměti dostupné pouze z privilegovaného režimu. Následně se konečně pokusí provést operaci, kterou požadovala aplikace – získat přístup k zadanému procesu.

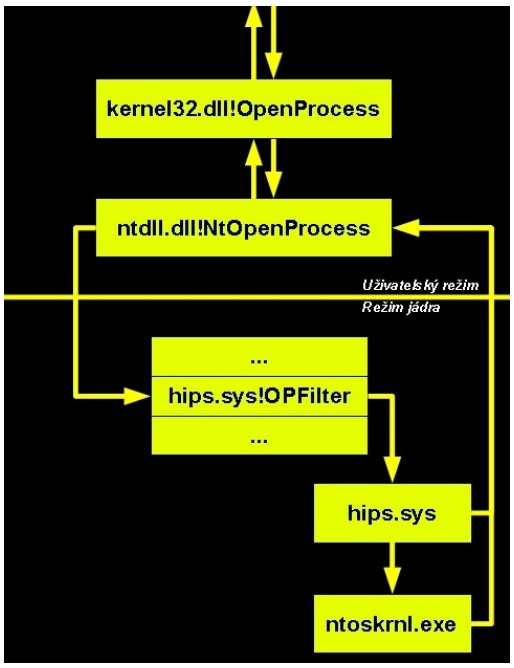


Obrázek 2: Průběh systémového volání na čistém systému

Na obrázku 3 vidíte situaci, kdy se systém HIPS pokusil převzít kontrolu nad přístupy k běžícím procesům pomocí modifikace obsahu tabulky systémových volání. Adresu rutiny *NtOpenProcess* nahradil adresou svého vlastního podprogramu *OPFilter*. Jádro operačního systému tedy při požadavku na přístup k nějakému procesu předá řízení rutině systému HIPS.

Rutina *OPFilter* se zpočátku chová podobně jako rutina původní; ověří platnost argumentů a zkopíruje jejich obsah do paměti přístupné pouze z privilegovaného režimu. Následně se systém HIPS rozhodne, zda přístup k danému procesu povolí či zablokuje. Pokud se rozhodne pro

blokování, nezavolá původní rutinu, ale vrátí chybový návratový kód, který bude předán aplikaci do uživatelského režimu. Pokud se rozhodne operaci povolit, předá řízení rutině *NtOpenProcess*.



Obrázek 3: Průběh systémového volání za účasti systému HIPS

KHOBE

Původní rutina je volána systémem HIPS úplně stejným způsobem, jako by ji volalo jádro systému na základě její přítomnosti v tabulce systémových volání. Neví tedy nic o tom, že argumenty pro operaci byly již překopírovány do paměti jádra. Z tohoto důvodu sama provede ověřování a kopírování do paměti jádra znovu a pokusí se získat přístup k procesu pomocí obsahu argumentů ze své kopie.

K ověřování platnosti argumentů a jejich kopírování do paměti jádra tedy dochází dvakrát. Jednou, aby systém HIPS mohl provést své rozhodnutí, podruhé za účelem provedení dané operace. Ale aplikace může obsah argumentů mezi těmito dvěma událostmi změnit!

Tato myšlenka (společně s útokem, na který vede) byla teoreticky popsána již v roce 1996 a tu a tam se objevila v dalších letech (2003, 2007, 2010) pod různými názvy (TOCTOU, argument switch attack, KHOBE).

V roce 2010 došlo pravděpodobně zatím k největšímu pozdvižení kolem tohoto problému, jelikož bylo prokázáno, že jím trpí drtivá většina tehdy používaných bezpečnostních balíků. Aktuální stav mi není znám. Obecně je však téměř nemožné prokázat, že daný software tímto problémem netrpí, nemáme-li k dispozici jeho zdrojový kód. Pokud ale vím, tato zranitelnost nebyla zneužita žádným malware vyskytujícím se ITW (in the wild).

Problém se vyskytuje téměř výhradně pouze u systémů, jejichž ochrana je založena na modifikacích kódu či modifikaci tabulek systémových volání. V dnešní době kvůli technologii Patchguard takových systémů ubývá, takže se zranitelnost KHOBE dostala opět relativně rychle do pozadí. Může však dojít opět k jejímu znovuobjevení. Zatím totiž neexistují způsoby, jak systémem HIPS chránit grafické uživatelské rozhraní (detekce instalace keyloggerů, snímačů obrazovky, útoků na grafické rozhraní aplikací) kromě modifikací kódu či tabulek systémových volání. A ovladač win32k.sys, zodpovědný za tyto operace, zatím není technologií Patchguard chráněn, čehož bezpečnostní balíky například od společnosti Avast, Kaspersky či Comodo využívají.

Tímto zakončíme povídání o technice modifikace tabulek systémových volání. Funguje podobně

jako modifikace kódu a disponuje i podobnými výhodami a nevýhodami. Hlavní plus spočívá v možnosti kontrolovat téměř každý aspekt života aplikací a snadná implementace, rozhodneme-li se KHOBE neřešit.

První nevýhoda opět spočívá v závislosti na architektuře procesoru. Mechanismus systémových volání je na 64bitových verzích Windows totiž implementován jinak než v případě 32bitové verze. Navíc, na 64bitových Windows lze bezpečně modifikovat pouze tabulku systémových volání zodpovědnou za operace s grafickým uživatelským rozhraním, zbytek hlídá Patchguard.

A poslední nevýhoda tkví v nutnosti implementovat vlastní ověřování argumentů a jejich kopírování do paměti jádra, což není úkol neřešitelný, ale může být nečekaně obtížný, a chránit se proti zranitelnosti KHOBE.

Direct Kernel Object Hooking (DKOH)

Nyní se budeme věnovat technikám, které dovolují kontrolovat jen určitý druh operací, ale dokáží tak činit jednodušeji (pohodlněji, bezpečněji) než techniky předchozí. Nejprve je však nutné osvětlit další principy používané v jádře Windows, protože na nich tyto techniky přímo staví.

Objekty jádra a jejich vlastnosti

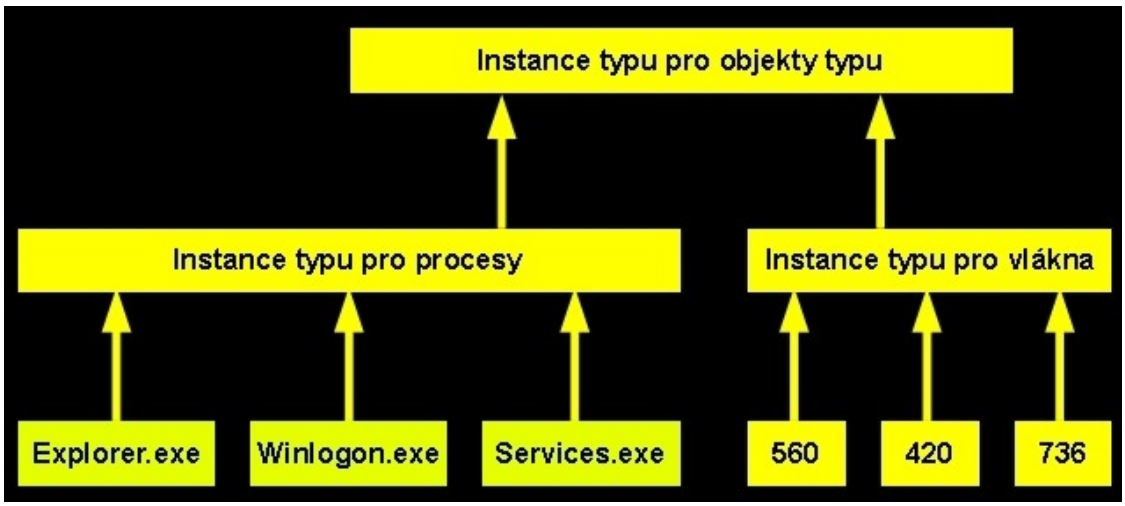
Jádro Windows se snaží různé entity (procesy, vlákna, otevřené soubory) reprezentovat jednotným způsobem. Jednotné rozhraní dovoluje s různými druhy entit provádět stejné operace. Rozdíly v implementaci těchto operací jsou skryty v nižších vrstvách kódu. Mezi tyto operaci patří pojmenování, nastavení oprávnění uživatelů k dané entitě i řízení přístupu. Každá entita, která je součástí tohoto rozhraní, je navenek označována jako *objekt jádra* (kernel object) či *objekt exekutivy* (executive object). První označení se používá hlavně v dokumentaci rozhraní Windows API (rozhraní dostupné v uživatelském režimu), druhé v nápovědě k balíku WDK (Windows Driver Kit), který dovoluje vytvářet ovladače jádra.

Aplikace s objekty jádra pracují pomocí nepřímých odkazů – *handle*. Jádro operačního systému je z každého handle schopno vyčíst adresu cílového objektu a oprávnění, kterými k němu handle disponuje. Než aplikace začne s daným objektem jádra pracovat, požádá systém o vytvoření nového handle s příslušným oprávněním (chce-li násilně ukončit proces, požádá o vytvoření handle s oprávněním násilného ukončení). Obdržené handle pak používá při volání různých rutin pro práci s daným objektem. Jakmile s daným objektem již nepotřebuje déle pracovat, požádá o zrušení daného handle. Handle je tedy něco jako dočasně vydaná přístupová karta.

Každá entita má ve své struktuře uložen odkaz na *objekt typu*. Jedná se o speciální objekt jádra, nepřístupný aplikacím, který v sobě uchovává různé vlastnosti a statistiky společné pro entity určitého typu. Mezi tyto údaje patří například to, kolik každá entita zabírá paměti, zda ji lze pojmenovat či kolik jich aktuálně existuje v systému). Pro každý druh entit existuje jedna instance objektu typu.

Ukázku propojení entit s instancemi objektu typu vidíte na obrázku 4. Tři entity reprezentující tři procesy (winlogon.exe, explorer.exe a services.exe) mají ve svých strukturách odkaz na instance objektu typu pro procesy. Obdobně, v pravé části obrázku vidíte tři vlákna, která v sobě nesou odkaz na instanci objektu typu pro vlákna. Obecně každá entita procesu odkazuje na objekt typu pro procesy a každá entita vlákna odkazuje na objekt typu pro vlákna. U ostatních druhů objektů jádra platí obdobné pravidla – vždy odkazují na instanci objektu typu svého druhu.

Jelikož objekty typu také patří mezi objekty jádra, i ony ve svých strukturách uchovávají odkaz na instanci objektu typu, která obsahuje charakteristiky společné pro jejich druh.



Obrázek 4: Entity a objekty typu

DKOH

Ve struktuře objektu typu jsou uloženy i adresy rutin, kterým jádro systému předá řízení během vykonávání některých operací. Každý druh entit může používat jiných rutin. Tento koncept dovoluje skrýt některé rozdíly mezi jednotlivými druhy entit pod společnou střechu jednotného rozhraní. Podobnost s virtuálními metodami z OOP jazyků není čistě náhodná.

Mezi operace, při kterých dochází k vykonávání těchto metod, patří:

- Vytvoření nového handle
- Zrušení existujícího handle
- Změna či čtení informací o oprávnění
- Získání jména

Některé metody mohou příslušnou operaci pouze monitorovat, jiné (vytvoření a zrušení handle) v případě potřeby i zablokovat. Systému HIPS tak stačí modifikovat adresy některých metod, aby získal kontrolu například nad přístupem k určitému druhu entit (procesům, vláknům). Tato technika se někdy označuje jako Direct Kernel Object Hooking (DKOH).

Hlavní výhodou DKOH oproti modifikacím kódu a modifikaci tabulek systémových volání je fakt, že k volání virtuálních metod dochází až po ověření platnosti argumentů a jejich zkopírování do paměti jádra. Není tedy potřeba provádět vlastní implementaci ověřování a kopírování a technika není zranitelná ani proti útoku KHOBE.

První problém spočívá v tom, že definice jednotlivých metod (argumentů, které dostávají) se často mění v závislosti na konkrétní verzi operačního systému, což zvyšuje složitost samotného systému HIPS. Další nevýhoda opět plyne z faktu, že se jedná o techniku oficiálně nepodporovanou; během vykonávání metody nemusí být možné počkat na rozhodnutí uživatele kvůli nebezpečí deadlocku. Tento problém odpadá, pokud se HIPS dokáže vždy rozhodnout automaticky. Třetí problém spočívá v tom, že datové struktury objektů typu (nebo jejich části) jsou hlídány technologií Patchguard na 64bitových verzích Windows.

OB Filtering Model

Windows Vista SP1 s sebou přináší rozhraní s názvem OB Filtering Model, které lze považovat za nadstavbu nad technikou DKOH. Jelikož je toto rozhraní podporováno přímo operačním systémem, odpadají problémy spojené s rozdíly na různých verzích systému a vzniká také nezávislost na

architektuře, protože Patchguard je ze hry.

Oproti DKOH však OB Filtering Model přináší relativně vysoká omezení. Předně, lze kontrolovat přístup pouze k procesům a vláknům. Kontrolovat či monitorovat lze pouze operaci vytváření nového handle. Pod pojmem „kontrolovat“ si ale nepředstavujte „blokovat za každých okolností“. Rozhraní vám umožní zjistit oprávnění nového handle a některá z nich tiše odebrat. Ne však všechna. Například oprávnění na čtení stavu procesu (či jeho paměti) odebrat nemůžete.

Rozhraní je implementováno tak, že ovladač předá jádru systému adresu rutiny, která má být volána v případě, že se někdo pokusí získat přístup k entitě daného druhu. Tato rutina pak může tuto operaci pozměnit, v některých případech i zablokovat. Dokonce může i počkat na reakci uživatele, zvláště v případě, kdy subjekt žádá o oprávnění, která mu lze zakázat. I tak je ale třeba velké opatrnosti.

Implementace rozhraní je provedena genericky, tedy fakt, že jej lze použít pouze na procesy a vlákna není natvrdo zadrátován v hlavním modulu jádra. Tato informace (včetně údaje o tom, jaká oprávnění lze blokovat) je uložena ve struktuře objektu typu reprezentující entity daného druhu. Lehkou modifikací struktur objektů typu lze tedy působnost rozhraní značně rozšířit. Otázkou zůstává, zda-li příslušnou část struktur typu nehlídá technologie Patchguard.

Zdroje a zajímavé odkazy

V této kapitole bych rád uvedl informace o zajímavých knihách, dokumentech a internetových stránkách, kde se dočtete podrobnější informace o tématech probíraných na mé přednášce.

Knihy obvykle neobsahují ty nejčerstvější informace, ale umožňují získat ucelený přehled o dané problematice, díky kterému pak můžete snadněji chápat aktuální dění. Rozhodl jsem se nabídnout následující kousky:

- **M. Russinovich, D. A. Solomon, A. Ionescu: Windows Internals: Including Windows Server 2008 and Windows Vista, Fifth Edition**. Kniha obsahuje velké množství informací o tom, jak Windows fungují „tam uvnitř“ a proč. Pokud o tomto OS chcete něco opravdu vědět, tak by tento titul neměl chybět ve vaší knihovničce. Myslím si, že mnoho informací bude pro vás srozumitelných, i když neovládáte žádný programovací jazyk. Zároveň kniha obsahuje množství tzv. experimentů, které dovolují nahlédnout pod pokličku reálně běžícímu systému. Čtvrté vydání knihy bylo přeloženo i do češtiny pod názvem Vnitřní architektura Microsoft Windows.
- **Martin Dráb: Jádro systému Windows**. Na počátku září roku 2011 po více jak dvou letech snažení se na trhu objevila moje první (a zatím poslední) kniha. Její zaměření je podobné jako u předchozího titulu, obsahuje ale také množství pasáží s obecnými informacemi o operačních systémech (popis různých algoritmů na řešení některých problémů) a v některých případech zabíhá až na úroveň programátorskou. Její záběr je však oproti Windows Internals značně užší. Knihu doprovází webové stránky, ze kterých si můžete stáhnout i zdrojové kódy ukázkových ovladačů jádra zkoumajících určitý aspekt operačního systému.
- **G. Hoglund, J. Butler: Rootkits: Subverting the Windows Kernel**. Tato, již letitá, kniha obsahuje pohled na bezpečností aplikace „z druhé strany barikády.“ Dozvíte se v ní, jak techniky, jako modifikace kódu, modifikace systémových volání a další, použít za účelem skrytí přítomnosti svého programu/ovladače před pátravým zrakem bezpečnostní aplikace. Jinými slovy: celá kniha pojednává o tom, jak programovat rootkity. Kniha sice neobsahuje žádné informace ohledně Windows Vista a novějších verzích operačního systému, ale myslím si, že její hlavní užitek spočívá ve vysvětlení různých technik skrývání přítomnosti, které se částečně prolínají i s technikami použitými pro implementaci systémů HIPS.

Dokumenty a zajímavé články:

- **Kernel Data and Filtering Support for Windows Server 2008, Microsoft Corporation**. V tomto dokumentu najdete informace, jak se na 64bitových platformách obejít bez modifikací kódu a tabulek systémových volání. Dočtete se například o tom, jak blokovat spuštění nových procesů a jak přesně funguje OB Filtering Model.
- **KHOBE – 8.0 earthquake for Windows desktop security software**. Článek o zranitelnosti TOCTOU (KHOBE) z roku 2010, který vzbudil pravděpodobně zatím největší pozornost.
- **Checking for Race Conditions in File Accesses**. Původní článek, ve kterém byla zranitelnost TOCTOU (KHOBE) poprvé představena.
- **Plague in (security) software drivers**. Článek pojednává o ověřování platnosti argumentů a jejich kopírování do paměti jádra těsně po systémovém volání a o tom, jaké s tím byly v minulosti potíže.
- **More on callbacks: ObRegisterCallbacks**. Povídání o tom, jak rozšířit funkčnost rozhraní OB Filtering Model i na jiné druhy objektů jádra než procesy a vlákna.

Zajímavé internetové stránky:

- http://www.kernelmode.info. Diskuzní fórum, na kterém se scházejí odborníci v oblasti programování ovladačů jádra, detekce rootkitů a implementace systémů HIPS.
- http://www.sysinternals.com. Na tomto serveru najdete spoustu utilit, které dovolují zkoumat různé aspekty operačního systému. Zajímavý je také blog M. Russinoviche. Dříve zde existovalo i velmi kvalitní fórum, jehož elita však přesídlila na kernelmode.info.
- http://www.jadro-windows.cz. Internetové stránky věnované mojí knize, kde krom blogu také naleznete zdrojové kódy ukázkových ovladačů a další materiály ke studiu.