Security-portal konference

Snímek 1: 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.

Snímek 2: 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.

Snímek 3: Na minulé konferenci jsme si povídali zejména o technikách implementace systémů HIPS využívaných ve velké míře v minulosti. Dnešní přednášku plánuji zaměřit zejména na 64bitové verze Windows a problémy, které pro programátory HIPS přinášejí.

Zavadíme také o rozhraní, která pro vývojáře připravil přímo Microsoft a která dovolují některé problémy řešit legitimní a dokumentovanou cestou. Tato rozhraní vznikla zejména jako náhrada za dříve používané techniky, jenž byly v jádře 64bitového systému zakázány.

Budeme mluvit také o některých mechanismech, jenž sice nebyly vytvořeny za účelem usnadnění implementace systémů HIIPS, ale i tak mají pro toto odvětví svůj význam. Během výkladu se budu muset opřít o některé principy fungováné operačního systému. Budu se ale vše snažit v dostatečné míře vysvětlit, abyste přednášce rozuměli i bez předchozích znalostí.

Snímek 4: Věci, o kterých budu konkrétně mluvit, vidíte na tomto snímku. Nejprve stručně popíšu technologii Kernel Patch Protection (Patchguard), jelikož se na 64bitových verzích Windows jedná o věc naprosto zásadní. Budu pokračovat rozhraním OB Filtering Model, jež lze využít na ochranu procesů a vláknem před útoky nežádoucích aplikací. Pro ty z vás, kteří se zúčastnili Hacking & Security Conference, to bude malé opakování.

Následně se podíváme na chráněné procesy, což je právě jeden z aspektů, který se neobjevil za účelem usnadnění práce programátorů systémů HIPS, ale z důvodu zcela jiného. Zbytek přednášky se budeme zabývat grafickým uživatelským rozhraním a možnostmi, které útočníkům přináší. Ve Windows totiž grafické rozhraní není jenom o kreslení a psaní do textových polí a klikání na tlačítka.

Snímek 5: Na tomto snímku vidíte poměrně zjednodušené schéma architektury operačního systému. Ze snímku 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.

Knihovna zakóduje svůj požadavek pomocí čísla, které v jednom z registrů procesu předá kódu jádra operačního systému. Jádro toto číslo použije jako klíč do tabulky systémových volání, zjistí adresu rutiny, která má danou operaci na starosti, a předá ji řízení Rutina se požadovanou operaci pokusí provést a její výsledek vrátí opět do uživatelského režimu. Na 32bitových verzích Windows je tabulka systémových volání reprezentována polem adres rutin a číslo operace se používá jako index.

Takováto struktura je dána tím, že jak kód aplikací, tak kód knihoven běží v *uživatelském režimu* procesoru, 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 zaručují doručování požadavků správným ovladačům. Existují dvě: jedna pro *ntoskrnl.exe*, druhá pro *win32k.sys*.

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.

Nyní tedy máte základní znalosti o tom, jak architektura systému vypadá a můžeme se podívat na první téma přednášky, kterým je technologie Kernel Patch Protection (Patchguard).

Snímek 6: Jejím hlavním úkolem je kontrolovat integritu některých důležitých součástí jádra operačního systému. Tato novinka představovala docela velký problém pro tvůrce bezpečnostního software, kteří si často zvykli implementovat ochranu právě pomocí modifikací kódu jádra a jeho důležitých datových struktur (PC Tools). Díky této události ale také vznikla oficiální rozhraní, kterými se Microsoft snaží nemožnost modifikací kódu nahradit. V opačném případě by nebylo možné některé problémy vůbec řešit (například ochrana procesů a vláken).

Podle několika různých zdrojů lze Patchguard definovat jako soubor rutin a cacheovaných kontrolních součtů či hashů obsahu kódu některých ovladačů jádra a důležitých datových struktur. Jednotlivé rutiny jsou vykonávány při různých, většinou náhodných, událostech, aby nebylo možné Patchguard snadno vypnout.

S tím souvisí i fakt, že se programátoři Microsoftu snažili Patchguard do jádra zakomponovat tak, aby ztížili reversním inženýrům práci. Například inicializace této technologie začíná dělením dvou zdánlivě nesmyslných čísel, které vyvolá výjimku procesoru pro dělení nulou.

Pokud Patchguard zjistí, že byly hlídané datové struktury či kód změněny, způsobí modrou obrazovku smrti, čímž zataví běh celého systému. V režimu ladění není tato technologie aktivní.

Snímek 7: Již jsme si řekli, že díky technologii Patchguard se objevilo několik nových rozhraní, která umožňují relativně snadno implementovat ochranu určitých oblastí. To ale neznamená, že se některá z nich neobjevila již dříve. Za příklad si můžeme vzít rozhraní na kontrolu registru či souborového systému.

Rozhraní OB Filtering Model (ochrana procesů a vláken) a Windows Filtering Platform (práce se sítí) se objevila až ve Windows Vista Service Pack 1. První verze WFP přišla spolu s Windows Vista SP0, ale jeho implementace obsahovala závažnou chybu, kvůli které se využití tohoto rozhraní pro účely blokování síťových spojení stalo prakticky nemožné.

Snímek 8: 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.

Snímek 9: Windows Vista SP1 s sebou přináší rozhraní s názvem OB Filtering Model. Jelikož je podporováno přímo operačním systémem, odpadají problémy spojené s rozdíly na různých verzích Windows a vzniká také nezávislost na architektuře, protože Patchguard je ze hry.

Oproti různým přímým modifikacím kódu a datových struktur objektů jádra 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 na čtení 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 příslušnou operaci pozměnit, což se v některých případech rovná zablokování. 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 společného pro entity stejného druhu. Lehkou modifikací struktur objektů typu lze tedy působnost rozhraní značně rozšířit. Otázkou zůstává, zda-li příslušnou část struktur typu nehlídá technologie Patchguard.

Snímek 10: Windows Vista s sebou také přináší speciální druh procesů – tzv. *chráněné procesy* (protected processes). Procesy tohoto druhu mají tu výhodu, že systém dovoluje obyčejným procesům a vláknům k těmto novým entitám (a k jejich vláknům) pouze velmi omezený přístup. Přičemž tato bariéra se nachází pouze mezi obyčejnými a chráněnými procesy, Přístup chráněných procesů k ostatním procesům a vláknům (i chráněným) funguje tak, jako by žádné chráněné entity neexistovaly.

Mezi další zajímavosti ohledně chráněných procesů patří i fakt, že tyto speciální procesy nemohou vytvářet synovské procesy.

Chráněný proces však nemůže vytvořit každý jouda. Jeho soubor musí být podepsán speciálním certifikátem, který podle dokumentaci zatím vlastní pouze Microsoft. Stejným způsobem musí být podepsány i knihovny, které chráněný proces používá. Přítomnost tohoto certifikátu je zřejmě kontrolována pouze při vzniku daného procesu.

Snímek 11: Na tomto snímku vidíte, jaké operace mohou s chráněnými procesy a jejich vlákny provádět obyčejné procesy. Vezmeme-li v potaz, že k procesu lze získat asi dvanáct různých oprávnění a ke vláknu jedenáct, je to opravdu málo. O chráněném procesu je možné zjistit některé informace, lze na něj čekat, násilně jej ukončit či pozastavit. Jeho vlákna můžete pozastavit, čekat na jejich ukončení, měnit a číst některé jejich vlastnosti. Informace o kontextu mezi ně ale nepatří.

Nyní možná začínáte chápat, proč se koncept chráněných procesů pro účely systémů HIPS tak úplně nehodí. Chráněné procesy lze totiž ukončit či pozastavit, jejich vlákna lze pozastavit. Obojí má pro procesy a vlákna systému HIPS fatální následky. Otázka tedy je: proč tento koncept existuje?

Snímek 12: Pravým důvodem vzniku (nebo alespoň důvodem uváděným v oficiálním dokumentu od Microsoftu) je lepší prostředí pro implementaci DRM. Jelikož chráněné procesy, na rozdíl od rozhraní OB Filtering Model, zakazují i čtení jejich adresového prostoru, mohou si vesele šifrovat a dešifrovat, co potřebují, aniž by je nějaká další aplikace obtěžovala.

Standardně se ve Windows objevují dva chráněné procesy: *System* a *audiodg.exe*. První slouží

převážně ovladačům jádra jako bariéra před aplikacemi, druhý má zřejmě opravdu něco společného s chráněným obsahem.

Ovladače jsou však velmi mocné, takže by se dalo předpokládat, že jejich pomocí půjde nějaký ten chráněný proces vytvořit. A opravdu, stačí změnit jediný bit v jeho struktuře. Změna má okamžitý účinek – operační systém se začne k procesu chovat jako k chráněnému, až samozřejmě na to, že neověřuje digitální podpisy jeho souboru a používaných knihoven.

Snímek 13: Tím se rozloučíme s procesy a vlákny a podíváme se na něco zcela jiného – na grafické uživatelské rozhraní. Služeb s ním spojeným lze totiž využít i k jiným účelům než ke kreslení.

Služby grafického uživatelského rozhraní jsou implementovány v ovladači *win32k.sys,* který obsahuje i vlastní tabulku systémových volání. Tato tabulka zajišťuje správnou distribuci požadavků aplikací v rámci ovladače.

Za ústřední prvek grafického rozhraní lze označit entitu zvanou *okno* (windows). Reprezentuje různé druhy ovládacích prvků (okna, tlačítka, textová pole, rolovací menu...). Zajímavé je, že okna nepatří mezi objekty exekutivy, a tedy příliš nepodléhají bezpečnostnímu modelu.

Sice se k nim také přistupuje přes handle, ale ta neplní úlohu přístupových karet, nýbrž jakéhosi jedinečného identifikátoru podobnému PID u procesů a TID v případě vláken. Aplikace mohou toto „handle“ uvádět při volání rozmanitých rutin, aniž by nějakou přístupovou kartu potřebovaly.

Celé fungování grafického uživatelského rozhraní je založeno na zasílání zpráv. Okna se tak dozvídají, co se s nimi děje, a mohou tak příslušně reagovat.

Zatím to tedy nevypadá, že by GUI představovalo příliš nebezpečnou zbraň v rukách malware. Počkejme si ale na další snímek.

Snímek 14: Pocit bezpečí je v tomto případě dost klamavý. Služby ovladače *win32k.sys* pro systémy HIPS opravdu zajímavé jsou. A autoři některých bezpečnostních produktů to dávají najevo.

Možnosti škodlivého kódu jsou různé. Asi předpokládáte, že malware může sledovat stav klávesnice, myši či obsah obrazovky. Ono je možné monitorovat (a ovlivňovat) doručování snad všech druhů odeslaných zpráv. Je také možné útočit na ostatní aplikace zasíláním různých (pro cíl neočekávaných) zpráv. Tím lze dosáhnout násilného ukončení cíle. O něco překvapivější je skutečnost, že služby ovladače *win32k.sys* dovolují injektovat kód do cizích adresových prostorů.

V současné době neexistují žádná legitimní a dokumentovaná rozhraní, jež tyto problémy dovolují jednoduše a účinně řešit. Tudíž si někteří výrobci bezpečnostního software pomáhají, jak umí: tedy modifikací kódu jádra a tabulky systémových volání ovladače *win32k.sys*.

Snímek 15: Paměť, kde je namapován ovladač *win32k.sys*, totiž není hlídána technologií Patchguard. To samé platí pro obsah jeho tabulky systémových volání. Z tohoto důvodu jsou staré dobré techniky implementace ochrany použitelné. Můžete si jich všimnout například v produktech firem Avast, Comodo či Kaspersky.

Některé problémy spojené se službami ovladače *win32k.sys* lze řešit i elegantnějšími způsoby, které modifikace jádra nevyžadují. Tyto postupy čerpají z toho, jak jsou jednotlivé mechanismy vnitřně implementovány.

Na 64bitových verzích Windows však ani použití starých dobrých modifikací kódu a datových struktur není tak jednoduché. Jako příklad si ukážeme modifikaci tabulky systémových volání. Na 32bitových verzích OS se jednalo o pole adres rutin zodpovědných za jednotlivé operace. Jádro prostě použilo číslo operace jako index, čímž bylo hledání adresy příslušné rutiny u konce.

Snímek 16: Na 64bitových Windows Vista a Windows 7 tabulka systémových volání obsahuje místo adres jednotlivých rutin pouze 32bitová čísla – offsety. Indexuje se do ní opět číslem operace, předávaným z uživatelského režimu v registru RAX. Samotný offset se skládá ze dvou částí. Spodní čtyři bity obsahují informaci o počtu argumentů rutiny zodpovědné za danou operaci, zbytek určuje

vzdálenost počátku této rutiny od počátku tabulky systémových volání.

Celá operace hledání adresy zodpovědné rutiny tedy probíhá tak, že se pomocí čísla operace zjistí offset, jenž se následně vydělí šestnácti (posun o čtyři bity doprava se znaménkem) a sečte se s adresou počátku tabulky systémových volání. Velikost offsetu, 28 bitů, dosti omezuje možnost přesměrování na ovladač systému HIPS (velká rozloha 64bit prostoru, ASLR). Obvykle se toto omezení obchází nepřímým přesměrováním; při svém startu ovladač systému HIPS najde v datech či kódu *win32k.sys* nevyužívanou část a zapíše do ní instrukci skoku. Na tu pak nasměruje offset v tabulce systémových volání. Problému se lze také vyhnout tím, že nebudeme měnit obsah tabulky systémových volání, ale přepíšeme počáteční instrukce požadovaných rutin.

Snímek 17: Pomocí těchto modifikací lze zabránit například ve škodlivém použití rozhraní Windows Hooks, které se poprvé objevilo snad již na 16bitových verzích Windows. Možnosti tohoto rozhraní jsou opravdu široké. Dovoluje totiž monitorovat (a ovlivňovat či blokovat) zprávy rozesílané jednotlivým oknům. Tím pádem umožňuje monitorovat i aktivity klávesnice a myší.

Další zajímavá vlastnost spočívá v tom, že monitorovací (či blokovací) kód je ve většině případů prováděn v kontextu procesu, jehož okno přijímá příslušnou zprávu. Proto je často nutné umisťovat tento kód do knihoven DLL, které systém v případě potřeby do ostatních procesů namapuje. K mapování dochází, až když je knihovny opravdu potřeba (tedy až v okamžiku přijímání zprávy daným oknem). I tak se jedná o jednu z možností, jak injektovat vlastní kód do cizích procesů.

Snímek 18: Avšak díky tomu, jak je rozhraní Windows Hooks implementováno, je možné tomuto způsobu injekce kódu a monitorování zabránit i bez modifikací jádra operačního systému.

Ovladač *win32k.sys* totiž při vykonávání kódu instalovaného pomocí Windows Hooks provádí něco podobného jako systémové volání, ale opačným směrem. Celou situaci ukazuje obrázek na tomto snímku. Ovladač donutí aplikaci provést určitý kus kódu v uživatelském režimu. Ovladač specifikuje číslo operace a adresu a velikost bloku paměti s parametry. Tyto údaje předá funkci *KeUserModeCallback* implementované v modulu ntoskrnl.exe. Tato funkce zajistí přechod do uživatelského režimu do rutiny *KiUserCallbackDispatcher* knihovny *ntdll.dll.* Během několika instrukcí dojde k nalezení pole *KernelCallbackTable* ve struktuře PEB, číslo operace je použito jako index. Tím je získána adresa rutiny zodpovědné za provedení operace požadované ovladačem. Adresy v poli *KernelCallbackTable* obvykle směřují buď do knihovny *user32.dll*, nebo do *wow64cpu.dll* (případ 32bitových aplikací emulovaných pod WOW64).

Struktura PEB obsahuje některé informace o procesu jako jsou aktuálně používané knihovny, aktuální adresář či hodnoty proměnných prostředí. Struktura jednak slouží jako vyrovnávací paměť (vlákna procesu kvůli každé informaci nemusí navštěvovat režim jádra), druhak obsahuje údaje, které jsou pro jádro systému nepotřebné, ale pro samotnou aplikaci velmi užitečné.

Jakmile je příslušná operace v uživatelském režimu vykonána, řízení je vráceno opět modulům *ntoskrnl.exe* a *win32k.sys* prostřednictvím rutiny *NtCallbackReturn* knihovny *ntdll.dll*.

Snímek 19: Aplikace se tedy může rozhodnout, zda monitorovací kód do svého adresového prostoru vůbec dovolí injektovat. Tím zabrání, aby nad ní mělo rozhraní Windows Hooks kontrolu.

A protože *win32k.sys* používá stejný mechanismus i při implementaci dalších rozhraní, lze tímto způsobem zvládnout i jiná nebezpečí. Navíc si vystačíte jen s modifikacemi kódu v uživatelském režimu, pokud dokážete zajistit, že vám je jiná aplikace nedokáže přepsat.

Podobným způsobem jádro systému do jednotlivých aplikací propaguje výjimky a asynchronní volání procedury.