Zranitelnosti ovladačů jádra v praxi

Martin Dráb
martin.drab@email.cz

Obsah

- Ovladače a zařízení
- Virtuální paměť
- Komunikace s ovladači
- Útoky na chybné ošetřování vstupů
- Systémová volání
- Útok záměnou argumentů

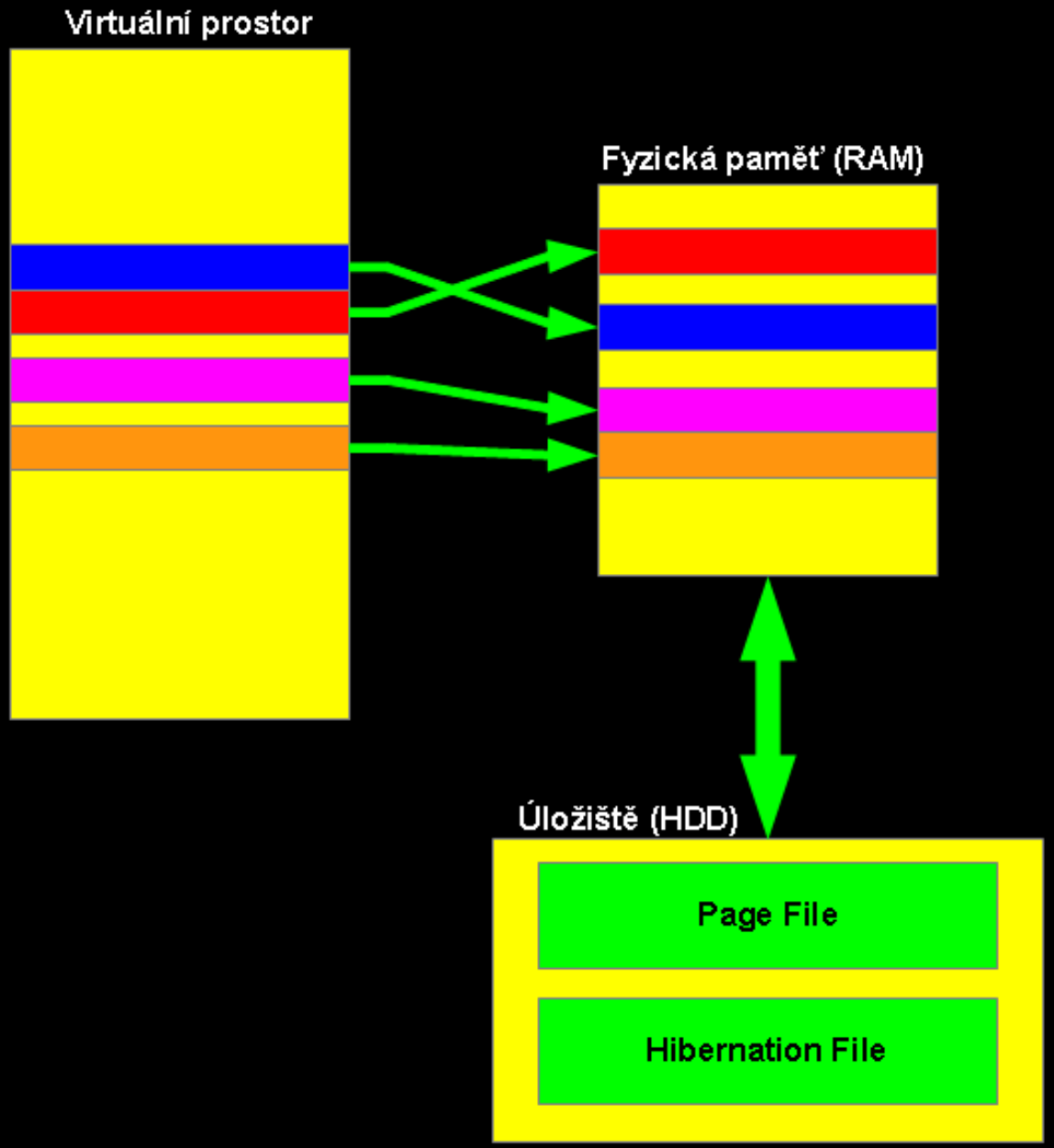
Ovladače

- Soubory formátu PE
- Vykonávány v režimu jádra
  - Plná kontrola nad daným strojem
- Speciální způsob spuštění
- Reprezentovány objektem **Driver**
- 64bitové verze Windows přidávají jistá omezení
  - Soubor ovladače musí být digitálně podepsán důvěryhodnou autoritou
  - Patchguard (ochrana proti modifikaci důležitých součástí jádra)

Zařízení

- Reprezentovány objektey **Device**
- Každé zařízení musí mít svůj ovladač
- Aplikace vidí zařízení jako soubory se speciálními jmény a mohou s nimi provádět různé operace (souborové)
- Veškeré požadavky na určité zařízení se zasílají k jemu příslušnému ovladači (metoda zpětně volané funkce – callback)

Virtuální paměť



Adresový prostor procesu

Komunikace s ovladači

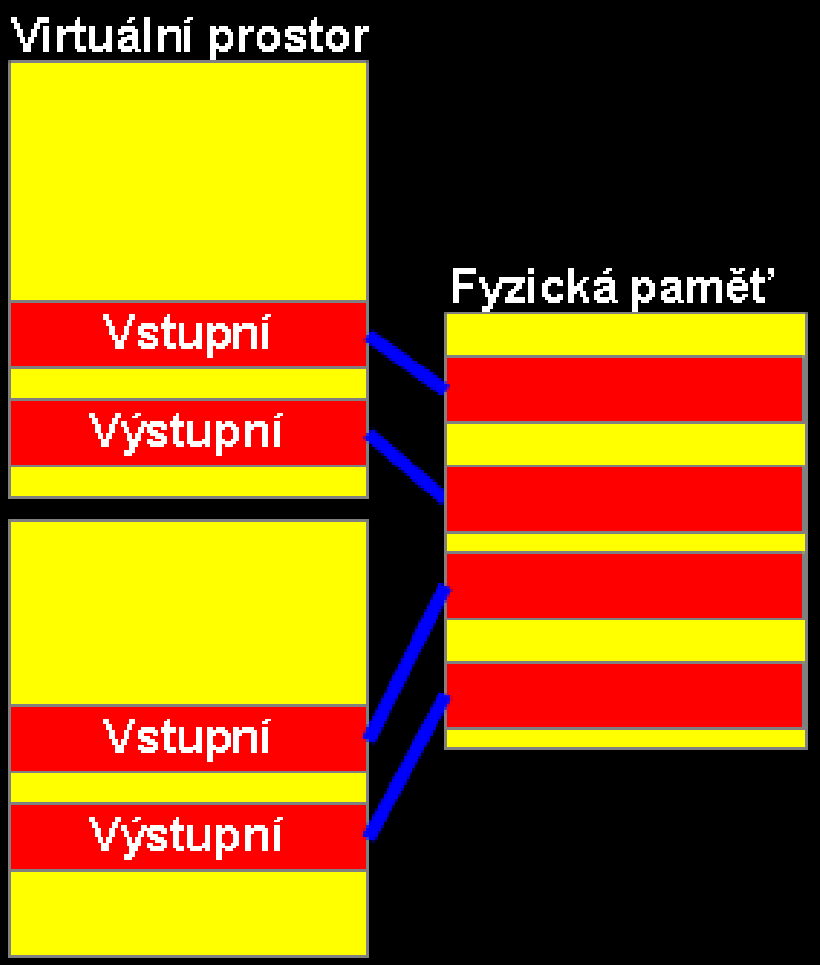
- Aplikace posílá na zařízení požadavky různých typů (čtení, zápis, obecná zpráva...)
- **Obecná zpráva:** kód požadavku, vstupní buffer, výstupní buffer
- U čtení/zápisu se udává pouze výstupní/vstupní buffer
- Vstupní i výstupní buffer musí aplikace specifikovat v rámci jí přístupné oblasti adresového prostoru
- Aplikace tedy může s touto pamětí manipulovat (uvolnit, změnit oprávnění, přepsat obsah)

Automatické řešení problému

- V rámci kódu obecné zprávy lze systému sdělit, aby vstupní a výstupní buffer automaticky překopíroval do paměti jádra.
- Aplikace nemůže ovladači „pod rukama“ paměť vstupního/výstupního bufferu uvolnit, změnit oprávnění či přepsat obsah.
- Systém buffery kopíruje do nestránkované paměti.
- Velké buffery = pomalé a paměťově náročné
- Neporadí si s „neplacatými“ buffery

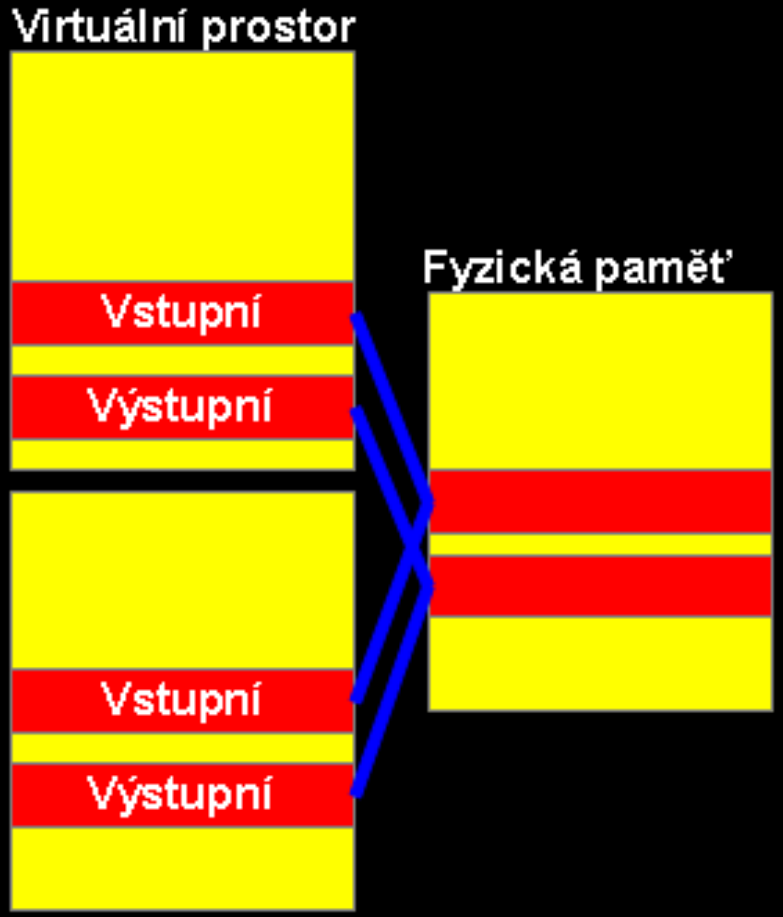
Manuální řešení #1

- Kopírovat si potřebná data sám
  - Ověřit, že:
    - Buffery neleží v paměti jádra
    - Lze z nich číst/do nich zapisovat
  - Kopírovat v try/except bloku
  - Aplikace udává adresy bufferů
  - Aplikace má plnou kontrolu nad jejich virtuálními stránkami



Manuální řešení #2

- Přemapovat buffery do paměti jádra
- Aplikace stále může obsah bufferů přepsat, nemůže ale paměť uvolnit „pod rukama“ či změnit oprávnění virtuálních stránek
- Granularita: velikost stránky
- Pro menší data pomalé



Důsledky špatného ošetření

- Modrá obrazovka smrti
- Možnost zapsat libovolná data na libovolné místo v paměti jádra
  - Lze využít k vykonání libovolného kódu v režimu jádra

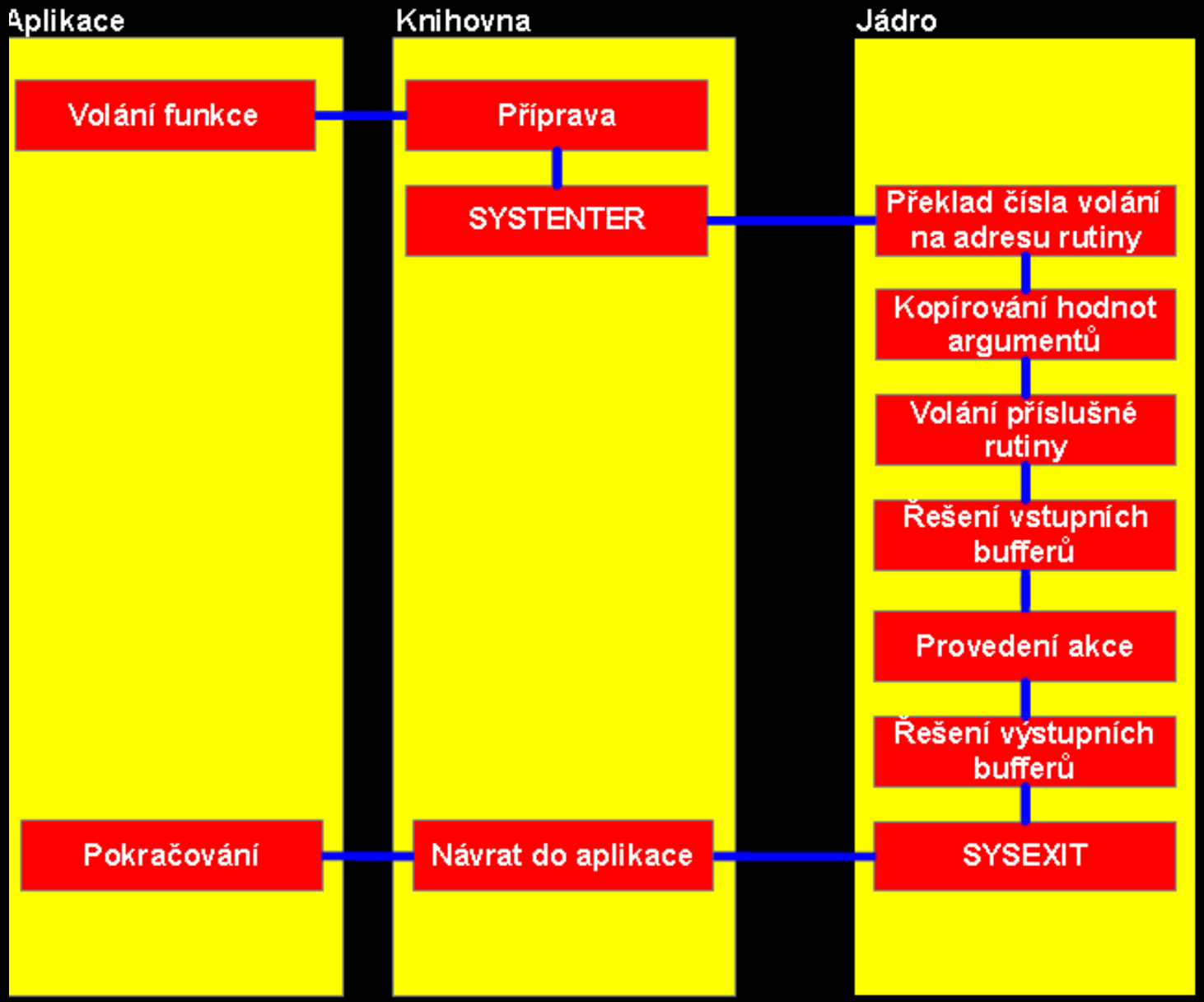
Praktická ukázka

- Ovladač dcr.sys (DriverCrypt)
- Při obsluze požadavku nekontroluje ukazatel předaný v rámci vstupního bufferu.
- Na adresu udanou v tomto ukazateli zapisuje data
- Aplikace tedy může zajistit vykonání svého kódu
  - Přepis položky **hal!HalDispatchTable**
  - **NtQueryInformationProfile** – zajistí vykonání funkce uložené v dané položce v **HalDispatchTable**

Systémová volání

- Speciální typ „volání“ podprogramu, který dovoluje aplikaci vykonat některou ze systémem definovaných rutin v režimu jádra.
- Místo adresy se používá číslo volání
- Jádro systému disponuje tabulkou, která čísla volání překládá na adresy rutin
- Hodnoty argumentů volání jsou překopírovány do paměti jádra
  - Celé buffeery ne, jádro pro každé sys. volání zná počet argumentů, ale ne jejich význam
  - Stejné problémy s buffery jako v minulém případě

Systémová volání II



Filtrování systémových volání

- Postup
  - Zkontrolovat argumenty
    - Dochází ke kopírování bufferů do paměti jádra
  - Dojít k rozhodnutí
  - Zavolat originální rutinu
    - Originální argumenty, použijí se buffery přístupné z aplikace
    - Obsah bufferů se od kontroly v prvním kroku mohl změnit!

Útok záměnou argumentů

- Zvolit cíl útoku (konkrétní systémové volání)
- Naplnit buffery tak, aby nedošlo k zablokování volání bezpečnostní aplikací
- Provést systémové volán
- Ve vhodný okamžik (bezpečnostní software volání povolil, originální rutina ještě nebyla volána) změnit obsah bufferů daty, která instruují originální rutinu k provedení nepovolené operace
- Důsledek: ukončení chráněných procesů, zápis do chráněných klíčů registru...

Praktická ukázka

- Norman Security Suite filtruje **NtTerminateProcess**
- **NtTerminateProcess** vyžaduje specifikovat „přístupovou kartu“ /handle) k cílovému procesu
- Průběh útoku:
  - Získat handle ke svému procesu (**OpenProcess**)
  - Použít jej při volání **NtTerminateProcess**
  - Ve vhodný okamžik handle zrušit (**CloseHandle**)

    a získat handle na cílvoý proces (**OpenProcess**)
  - S trochou štěstí budou mít obě handle stejnou hodnotu

Další zajímavosti

- Obecná chyba
  - Nezávislá na platformě
  - Často tam, kde se modifikuje kód
- Teoreticky známa jíž 17 let
  - 1996 – popsána na Unixu a velmi teoreticky
  - 2003/4, 2008, 2010
- Relativně „zastaralá“
  - 64bitové verze Windows stále používanější
  - Patchguard výrazně omezují modifikace kódu v jádře

Obrana

- Používat dokumentovaná rozhraní
- Modifikovat kód pouze tam, kde už nedochází k přenosu bufferů (a handle) do paměti jádra
- Nemodifikovat kód, ale datové struktury
- Donutit originální obsluhy systémových volání brát argumenty z paměti jádra
  - Dochází k „drobné“ změně sémantiky
  - Neprobíhá verifikace proti bezpečnostnímu modelu

Další čtení

- **KHOBE – 8.0 Earthquake for Windows Desktop Security Software** (http://www.matousec.com/info/articles/khobe-8.0-earthquake-for-windows-desktop-security-software.php)
- **Plague in Security Software Drivers** (http://www.matousec.com/info/articles/plague-in-security-software-drivers.php)

???

- **Martin Dráb**
- **Email:** martin.drab@email.cz

Závěr