WSL: Basics

*unrooted

What is WSL?

* WSL = Windows Subsystem for Linux
* WSL consists of the following main components:
    * a session manager executed in user mode
    * lxss.sys and lxcore.sys, which are pico process providers (translate syscalls)
    * pico process is started whenever Linux program is started
* WSL2 is using virtualized Linux kernel on Hyper-V, which was customized by Microsoft to improve fs performance and eliminate syscall incompatibilities

Pico Processes

* associated with pico provider kernel-mode driver
* support in Windows kernel was implemented with two layers:
    * minimal processes
        * empty user-mode address space
    * pico processes
        * minimal process with associated kernel mode driver (pico provider)
* host OS doesn’t try to manage user-mode address space inside the pico process
* Windows kernel passes all sys-calls/exceptions from user-mode of pico process to a pico provider to handle
* Pico provider registers with Windows kernel at boot time and exchanges interfaces
    * e.g. function pointers for kernel to call when dispatching a user-mode sys-scall
    * e.g. kernel provides function pointers for creating pico processes/threads
* not actively used in WSL2 - interaction with the kernel is more direct as WSL2 introduced real Linux kernel running under Hyper-V
* note: when a program is run, Linux subsystem driver requests to Windows kernel to run the process and calls ZwCreateUserProcess

Sys-calls

- the flow:
  - marshalling parameters - moving parameters into CPU registers
  - making the syscall with a trap to transition to kernel mode to make the syscall
  - return from the system call with another special instruction to return to user mode
  - user mode checks the return value
- utilizing lxss.sys
- notes:
  - WSL implements Linux pipes separately, but still relies on NT functionalities for primitives (data structures, synchronization)
  - sched_yield maps one to one with ZwYieldExecution

File “Systems”

- Using Virtual File System to allow multiple file systems to co-exist
- VFS implements system calls for file system operations using data structures such as:
  - index nodes (inodes)
    - information about file system objects: file type, permissions, size, last modified etc.
  - directory entries
    - uses directory entry cache to represent the file system namespace
    - dentry’s are in memory, no physical store, contain a pointer to inode for the file
- Special file types: device files, sockets, symlinks

File “Systems”

* WSL must perform the following fs operations:
    * translate Linux fs operations into Windows kernel operations
    * provide “special” file “systems” (so-called ProcFs, DrvFs etc.)
    * provide access to Windows volumes
    * provide a place where Linux system files can operate normally, allowing for file permissions, symlinks etc.
    * using lxcore.sys
* WSL VFS file systems include:
    * VolFs - /, /root, /home
    * DrvFs - /mnt/c
    * TmpFs - /dev
    * ProcFs, SysFs etc. - /proc, /sys, etc.

What’s next?

- dir C:\Users\<username>\AppData\Local\Microsoft\WindowsApps
  - that’s where .exe is stored
- mapped under \\wsl$\<distro name>
  - direct access to WSL filesystems
- more details under HKCU:\Software\Microsoft\Windows\CurrentVersion\Lxss
  - ‘base path’ – that’s where .vhdx is stored
  - ‘version’ – whether it’s WSL1 or WSL2
  - ‘distributionName’ – self-explanatory
- note: exception is --system WSL distro (.vhd stored under C:\Program Files\WSL\system.vhd)

What’s next?

* wsl.exe -u root
    * auto-logins into root user without asking for password
* wsl.exe -e /mnt/c/Windows/System32/calc.exe
* bash.exe -c calc.exe
    * can be used as AWL bypass
    * (wsl.exe only) can be combined with -u root for arbitrary Linux command execution, as well as use -d to change the distro we’re using for execution

What’s next?

stay tuned™

thanks for your attention

*unrooted