GAINING VISIBILITY INTO LINUX BINARIES ON WINDOWS: DEFEND AND UNDERSTAND WSL

BLUEHAT 2016

ALEX IONESCU

@AIONESCU

BIO

Vice President of EDR Strategy at CrowdStrike, a security startup

Previously worked at Apple on iOS Core Platform Team

Co-author of *Windows Internals 5th, 6th* and now *7th Edition*

Reverse engineering NT since 2000 – lead kernel developer on ReactOS Project

Instructor of worldwide Windows Internals classes

Conference speaking:

- BlueHat 2016, Infiltrate 2015
- Blackhat 2016, 2015, 2013, 2008
- SyScan 2015-2012, NoSuchCon 2014-2013, Breakpoint 2012
- Recon 2016-2010, 2006

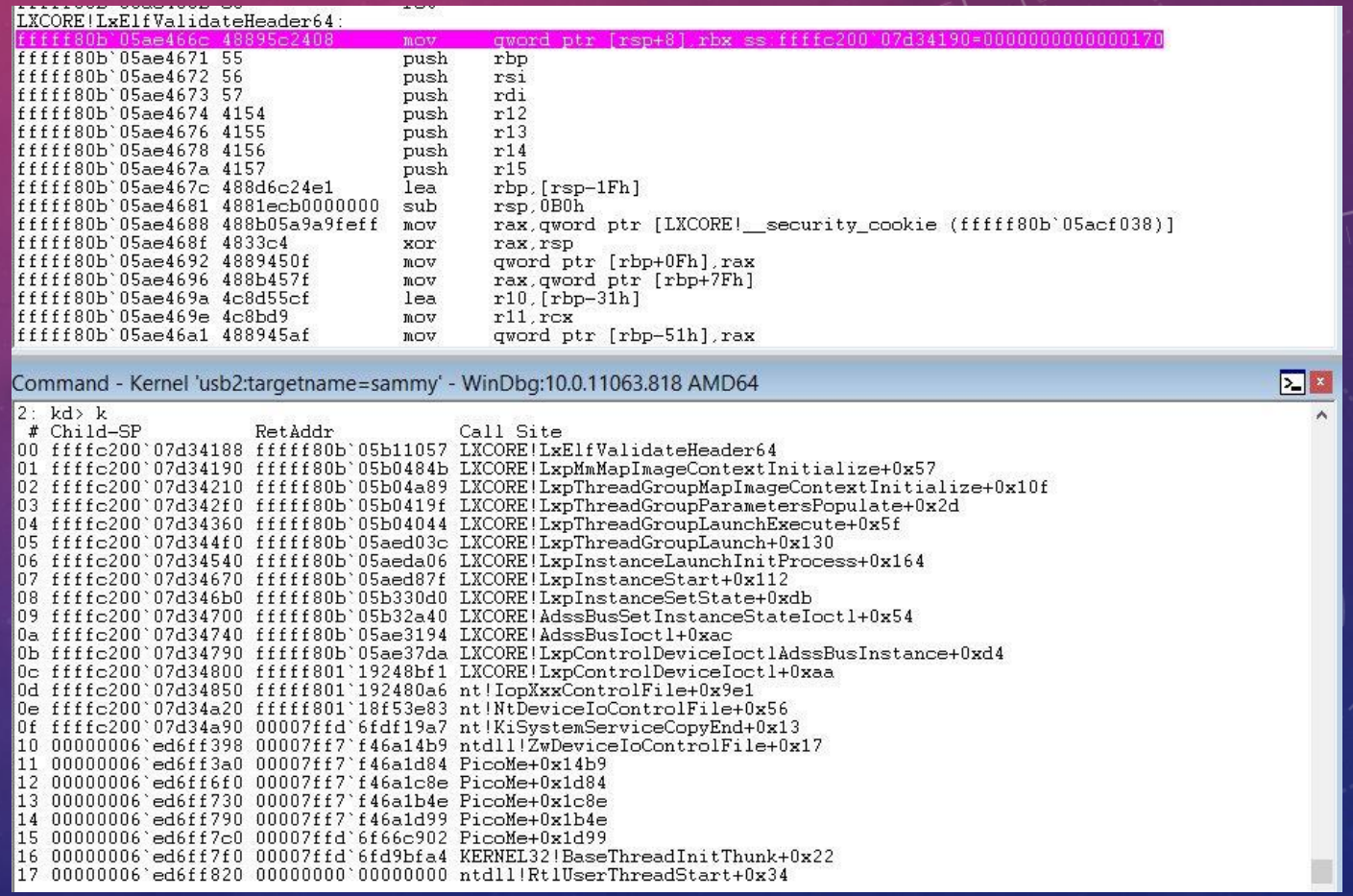
For more info, see www.alex-ionescu.com and www.windows-internals.com

INTRODUCTION

- I have been analyzing WSL since its first inception as “Astoria” and “Arcadia”
  - Based on Microsoft Research “DrawBridge” Project
- Achieved arbitrary ELF binary execution in January 2016 – no WSL installed (Redstone 1 Preview)
  - Discovered multiple design issues that would allow malicious users to abuse/exploit the subsystem
- Presented at BlackHat 2016 after most of the issues had been fixed in Anniversary Update
  - “Optionality” of subsystem was now enforced (driver no longer present by default at boot)
  - Protected Process boundary added to driver/subsystem, and Pico processes are isolated too
  - *Some* visibility issues addressed (*SeLocateProcessImageName* no longer returns NULL, for example)
- Many more visibility issues exist
- New challenges and features which can negatively impact security are coming in Redstone 2
  - Aka RS2 aka “Creator’s Update” aka 1703

DESIGN ISSUES IN RS1 PREVIEW BUILDS (FIXED)

- LxCore was installed **by default** in the kernel even if the feature is disabled and developer mode is turned off
- Checks were done only by LxssManager over the COM interface, but not the driver itself
  - Driver allowed Administrators to have RW Access
- As such, could completely bypass LxssManager/Developer Mode/Feature Installation and directly send commands to the driver (from Admin)
- Tweeted “PicoMe” in February before WSL was even announced



OUTLINE

- Architecture Review
  - Minimal and Pico Processes
  - Pico Provider Interface
  - LXSS Components
  - LXSS IPC Interfaces
- Changes and Updates in Redstone 2
- Visibility Issues and Security Flaws
- Call to Action
  - Passive Visibility (Auditing, Tracing, Forensics)
  - Debugging & Troubleshooting
  - Monitoring & Prevention (Antimalware, AppLocker)
  - SDL Process
- Q & A

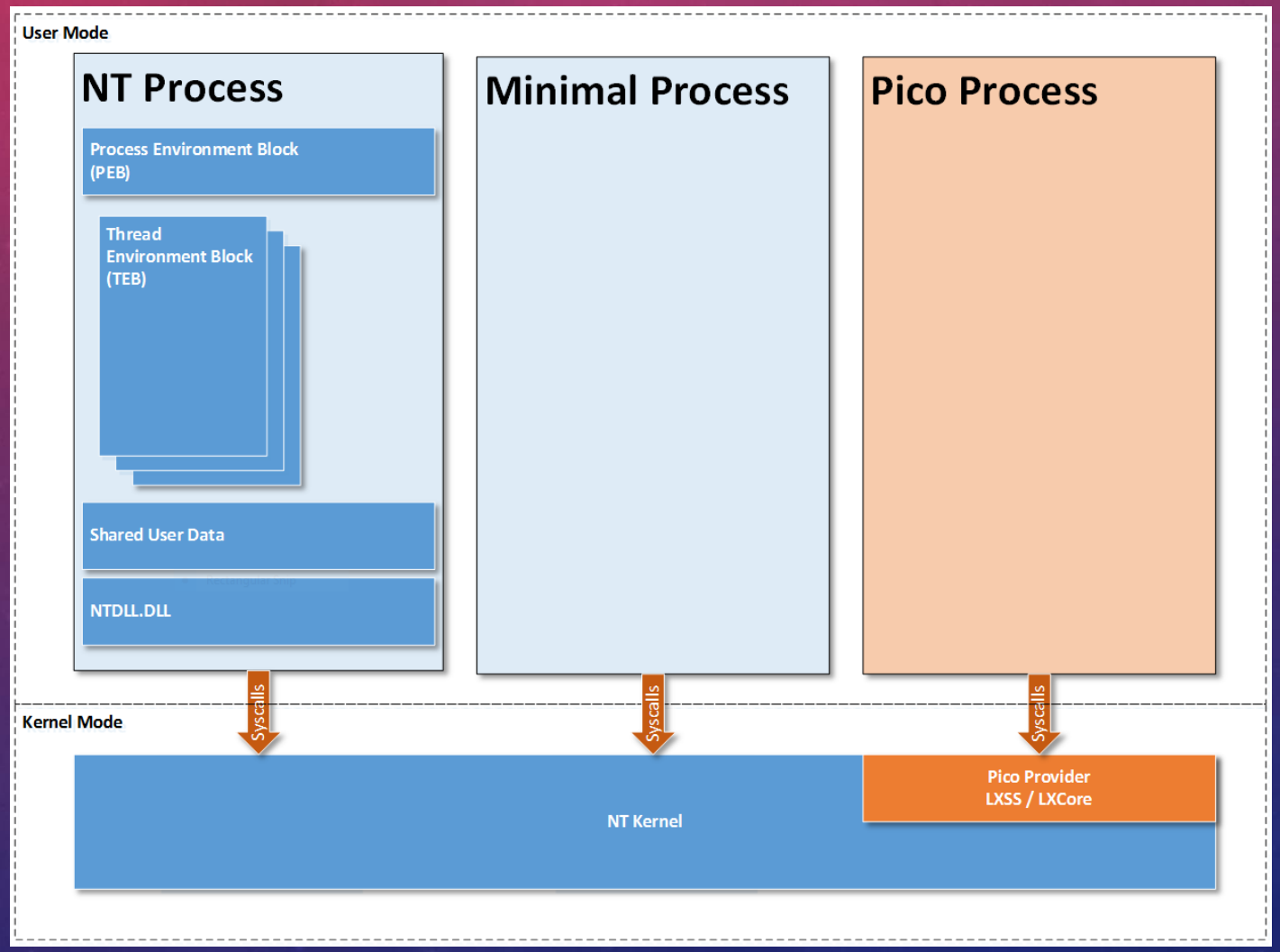
ARCHITECTURAL OVERVIEW

PICO PROCESS INTERNALS

PICO PROCESSES

- Minimal processes have no PEB, TEB, NTDLL, KSHARED_USER_DATA, NLS Tables, API Set Mappings, etc…
  - Can be created with *NtCreateUserProcess* with flag 0x800 if caller is Kernel Mode
  - Used by “Memory Compression” and “Secure System” processes
- A pico process is essentially a minimal process which has a provider attached to it
  - Provider is notified of system calls, exceptions, creation, termination, context changes, etc…
  - Provider determines access rights granted to non-pico processes
  - RS2: Provider indicates the type of Subsystem it implements
- Can only be created with special Pico interface that is provided to the registered Pico Provider
  - Not exported

BASIC BLOCK DIAGRAM



ref: https://blogs.msdn.microsoft.com/wsl/2016/05/23/pico-process-overview/

PICO PROVIDERS

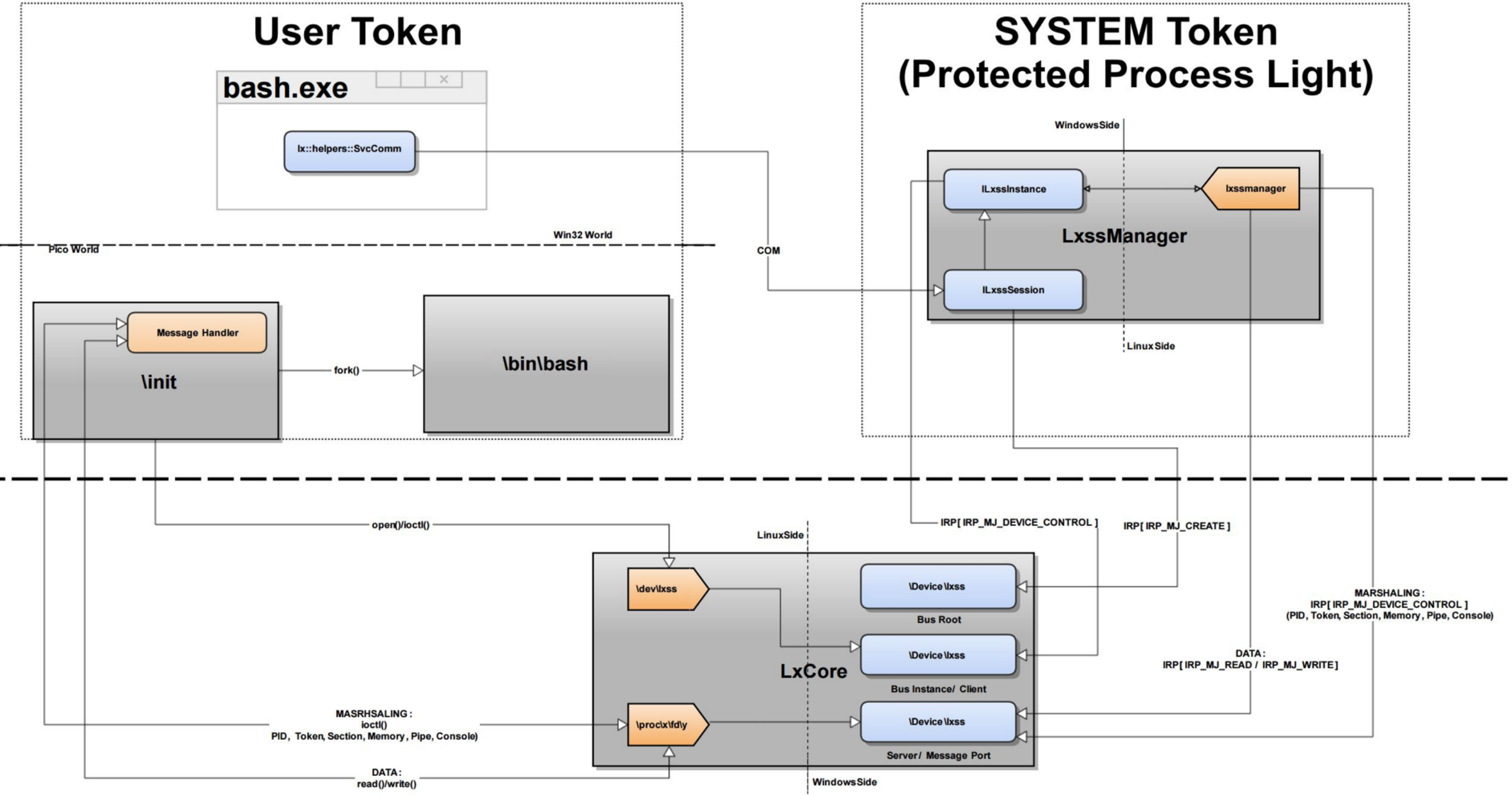
- Pico providers are essentially custom written kernel modules which implement the necessary callbacks to respond to the list of possible events (shown earlier) that a Pico process can cause to arise
  - NT implementation of the “Drawbridge” Microsoft Research Project
- Can be registered with *PsRegisterPicoProvider*
  - Pass in array of functions which handle the required event callbacks
  - Receive array of functions which allow creation and control of Pico processes (see next slide)
- However, only allowed if *PspPicoRegistrationDisabled* is FALSE
  - Set as soon as *PipInitializeCoreDriversByGroup* returns – before ELAM and 3rd party drivers can load
- Currently only one Pico provider can be registered, as the callbacks are simply a global function array

PICO PROVIDER SECURITY

- Pico Providers also “register” with PatchGuard, providing it with their internal list of system call handlers
- Essentially, this means that the Linux system calls are protected by PatchGuard, just like the NT ones
- Additionally, attempting to register a “fake” Pico Provider, or modifying key Pico Provider state will also result in PatchGuard’s wrath
  - For example, playing with *PspPicoRegistrationDisabled*
  - Even if restoring the value back to its original value
- Patching the callbacks (*PspPicoProviderRoutines*) will also result in detection by PatchGuard
- As seen before, only “core” drivers can be Pico Providers
  - Boot Loader enforces that core drivers are signed by Microsoft, and “lxss.sys” is hardcoded inside the code

WSL COMPONENT OVERVIEW

- A Pico Provider driver (LXSS.SYS / LXCORE.SYS) which provides the kernel-mode implementation of a Linux-compatible Kernel ABI and API, as well as a Device Object (\Device\lxss) for command & control.
  - It also provides a “bus” between the various WSL Instances and the NT world, and implements a virtual file system (VFS) interface, including a device inode for command & control on the Linux side.
- A user-mode management service (LxssManager), which provides an external-facing COM interface, relying information to the Pico Provider through its Device Object
- A Linux “init” daemon (like Upstart, systemd, etc…), which is created for every active WSL Instance and is the ancestor of all other Linux processes
- A Windows management process for performing updates/servicing, installation, uninstallation, and user management (LxRun.exe)
- A Windows launcher service which communicates with the LxssManager and spawns a new instance of WSL (and hence init daemon) and executes the desired process (typically “/bin/bash”)


User Token
bash.exe
lx::helpers::SvcComm
SYSTEM Token
(Protected Process Light)
WindowsSide
ILxssInstance
lxssmanager
LxssManager
ILxssSession
LinuxSide
Win32 World
Pico World
COM
Message Handler
\init
fork()
\bin\bash
open()/ioctl()
IRP[IRP_MJ_DEVICE_CONTROL]
IRP[IRP_MJ_CREATE]
LinuxSide
\dev\lxss
\Device\lxss
Bus Root
\Device\lxss
LxCore
Bus Instance/ Client
\proc\x\fd\y
\Device\lxss
Server/ Message Port
MARSHALING :
IRP[IRP_MJ_DEVICE_CONTROL]
(PID, Token, Section, Memory, Pipe, Console)
DATA :
IRP[IRP_MJ_READ / IRP_MJ_WRITE]
MASRHSALING :
ioctl()
PID, Token, Section, Memory, Pipe, Console)
DATA :
read()/write()
WindowsSide

ARCHITECTURAL OVERVIEW

LXCORE (KERNEL-MODE) INTERNALS

LXCORE FUNCTIONALITY

- LXCORE.SYS is the large (800KB) kernel-mode Ring 0 driver which implements all the functionality that a Linux application inside of a Pico process will see
  - In some cases, functionality is implemented from scratch (such as pipes)
  - In other cases, functionality is wrapped on top of an existing NT kernel mechanism or subsystem (scheduling)
  - In yet other cases, functionality is heavily built on top of an existing NT kernel primitive, with some from-scratch functionality on top (VFS + inodes on top of NTFS for VolFs/DrvFs, but on top of NT Process APIs for ProcFs, for example)
- Decision on how far to go down with wrapping vs. re-implementing was done based on compatibility
  - For example, Linux pipes have subtle enough differences from NT pipes that NPFS could not be used
- In some cases, compatibility isn't perfect – NTFS is not quite identical to EXT+VFS, but "close enough"
- Reliance on many key portability design decisions done by NT team from day 1 (for POSIX support)

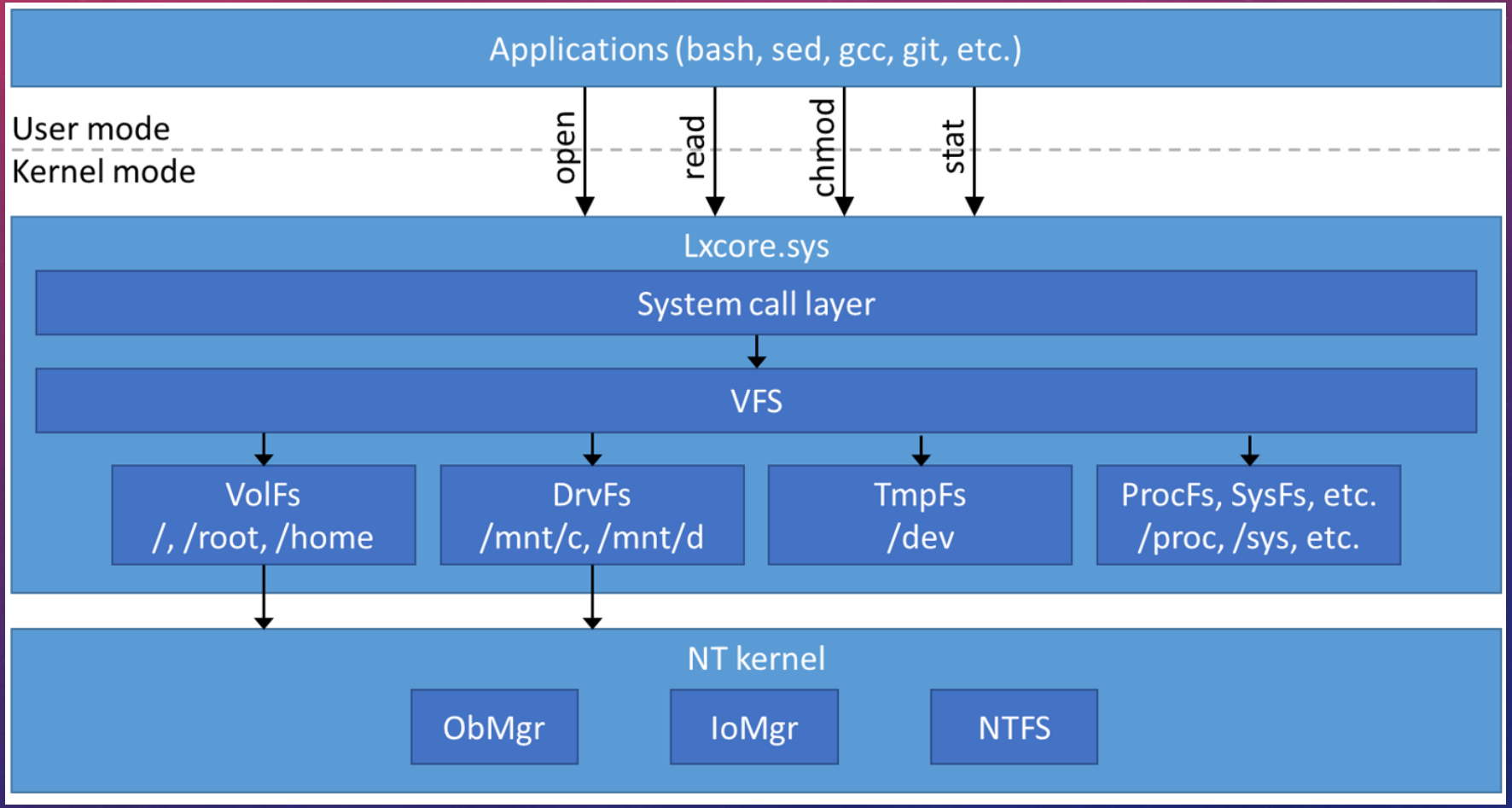
SYSTEM CALLS

- Full “official” list available from Microsoft: https://msdn.microsoft.com/en-us/commandline/wsl/release_notes
- 220 implemented as of the latest build
- Called by *LxpSysDispatch*
- Support for ptrace() tracing of system calls
- Can enable breaking on system call failure, for debugging
- *LxpTraceSyscall* used for both ETW logging (LxpPrintSysLevel) and DbgView (LxpTraceLastSyscall)

VFS / FILE SYSTEM

- Implements a “LxFs” system, which is the Linux-facing file system with support for UNIX rights, etc.
  - Implemented on top of Alternate Data Streams and Extended Attributes part of NTFS
  - Not interoperable with Win32 applications
- Also implements a “DrvFs” system, which is the Linux-facing file system that maps Win32 drives
  - Implemented as mount points that map to actual NTFS data on the machine
  - Rights are dependent on the token that was used to launch the initial WSL instance
- Provides emulation/wrapping/actual implementation for various virtual file system concepts:
  - ProcFs mostly works (calls kernel query APIs as needed) or just already has the data
  - TmpFs also exists and implements various devices
  - SysFs is there, etc…

VFS BLOCK DIAGRAM



ref: https://blogs.msdn.microsoft.com/wsl/2016/06/15/wsl-file-system-support/

DEVICE OBJECT INTERFACES

- Although \Device\lxss is a single Device Object, it implements 6 different interfaces on top of it.
- A Root Bus Interface – used by LxssManager to create new Instances. Only accepts the single IOCTL.
- A Bus Instance Interface – automatically opened by the Root Bus Interface when a new Instance is created. Internally represented as \Device\lxss\{Instance-GUID}. Acts as the command and control interface for the Instance.
- A Bus Client Interface – used by LxssManager whenever an IPC Bus Server is registered or connected to, such as when talking with the init daemon. Represented as \Device\lxss\{Instance-GUID}\Client
- An IPC Server Port Interface – used when an IPC Bus Server is registered. Represented by \Device\lxss\{Instance-GUID}\ServerPort
- An IPC Message Port Interface – used when an IPC Bus Client connects to its Server. Represented by \Device\lxss\{Instance-GUID}\MessagePort
- NEW in RS2: A VFS File Interface – used when marshalling VFS files from Linux to Win32. Represented by \Device\lxss\{Instance-GUID}\VfsFile

LXSSMANAGER INITIALIZATION

- LxssManager (LXSSMANAGER.DLL) is a service-hosted service living inside of a SVCHOST.EXE process
- Runs as a Protected Process Light (PPL) at the Windows Level (0x51)
- Registers a COM Class (CLSID_LxssUserSession) which implements the IID_ILxssSession interface

BASH.EXE INTERFACE

- BASH.EXE is responsible for being the Win32-friendly launcher of WSL Instances
- lx::bash::DoLaunch is used to either launch /bin/bash or another binary, if BASH.EXE –c “valid path” was used
  - NOTE: BASH.EXE –c “man 2 open” will actually launch “/usr/bin/man” and not “/bin/bash –c /usr/bin/man”
- Launch is done by using the same lx::helpers::SvcComm class as LXRUN.EXE
- Reads the DefaultUid in the user’s registry settings to associate the correct Linux user with which to spawn
  - HKCU\Software\Microsoft\Windows\CurrentVersion\Lxss\

LXSS IPC INTERFACES

WIN32-LINUX COMMUNICATION

SOCKETS / FILES

- Unix sockets are supported for Linux-Linux application communication, but Internet sockets are also implemented, allowing both local and remote communications over IP
- By creating a localhost server, a Win32 application can connect to it and engage in standard socket API communications
  - Similarly, the server could be on the Win32 side, and the client is in an WSL Instance
- UDP raw sockets are supported, but require an Admin Token on the NT side to comply with TCPIP.SYS checks
  - RS2: No longer the case for ICMP as specifically used by “ping”
- Using DrvFs, it should be possible for a Windows and Linux app to use read/write/epoll on a file descriptor on the Linux side, and a file object on the Windows side with Read/Write/WaitForSingleObject

BUS IPC (“ADSS” IPC) – LXBUS

- The only other way for a Windows and Linux application to communicate is to use the internal LxCore-provided “Bus” between clients of an instance, part of the original “ADSS” interface (Android Sub System)
- Allows a named server to be registered and connected to, after which both sides can:
  - Use Read/Write API calls for raw data
  - Use IOCTL API calls for marshal/unmarshal operations
- The Windows side needs to register a server by either sending the correct IOCTL (to the Bus Instance) or by using the ILxssSession COM Interface and calling the RegisterAdssBusServer method (Administrator only)
  - Then, can use the IOCTL_ADSS_IPC_SERVER_WAIT_FOR_CONNECTION on the returned Server Port handle
- The Linux side needs to open a handle to the Bus Instance (\dev\lxss) as root (UID 0) and then send the IOCTL_ADSS_CONNECT_SERVER IOCTL
  - Restricted to the init daemon only (PID must be 1)
  - Or, set “RootAdssbusAccess” in HKLM\CCS\Services\lxss\Parameters key to 1

NEW REDSTONE 2 (RS2) CHANGES

EXPANDING THE FEATURE SET

ANONYMOUS IPC SERVER

- IPC communication over LxBus was only possible from root Linux side and Admin Win32 side
  - Now, unprivileged COM client can spawn Linux process and receive an “unnamed” IPC server handle
  - Linux side can unmarshal by sending IOCTL to LxCore
- To support the above, new /dev/lxssclient interface is added, which does not require root rights
  - Allows marshalling of VFS files to Win32 side
  - Allows unmarshalling unnamed server handle
- In other words, unprivileged Linux and Win32 can now communicate over the LxBus, exposing IPC marshal/unmarshal bugs

KERNEL-DRIVEN CONSOLE I/O & PIPE INTEROP

- Before, Bash.exe owned its own console, and implemented its own loop to wait for input
  - This required creating a number of NT pipes, connecting to them from the Linux side, and marshalling the connected pipe handles through LxBus to the LxCore driver that owned the TTY
  - Then, Bash.exe had to handle character conversion and convert messages it received in its pipe to console output
- Responding to Windows-driven changes to the console (for example, resizing) left the Linux side out-of-sync
  - Bash.exe had some code to try to handle some of this, using yet another pipe to send messages to LxCore
  - Some issues unfixable through this mechanism
  - Also was hard to allow the Linux side to pipe stdout to a Windows file effectively, or vice-versa
- In Redstone 2, LxCore takes advantage of the Condrv.sys component that actually owns console logic since Windows 8.1 (originally developed for MinWin/WoA purposes)
  - LxCore.sys now has kernel-mode wrappers for Win32 Console API like *KernelReadConsoleInputExW*
    - Always comforting to see legacy APIs and “W” in kernel…

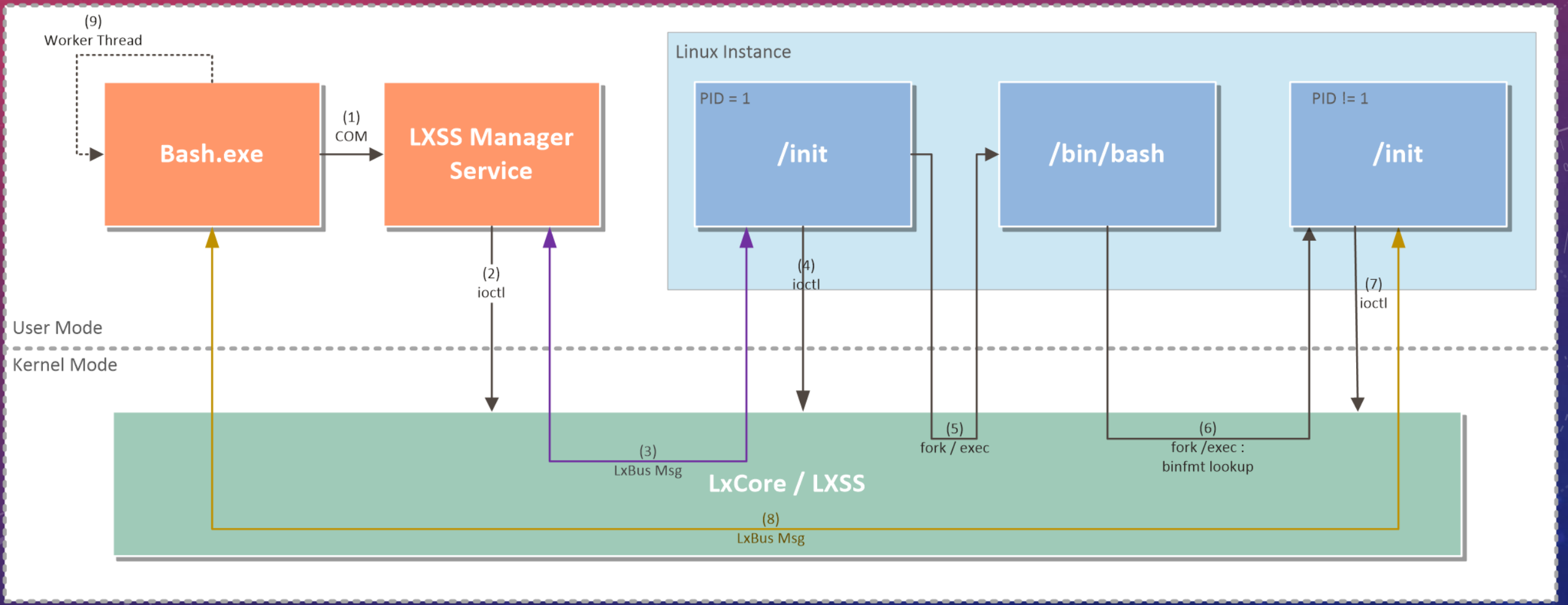
NEW/REMOVED IOCTLS

- The following IOCTLs have been added (so far) to Redstone 2
  - 0x2200BF = IOCTL_ADSS_IPC_NT_PROCESS_FILE_WAIT_FOR_SIGNAL
  - 0x2200B7/BB = IOCTL_ADSS_IPC_CONNECTION_MARSHAL/UNMARSHAL_VFS_FILE
  - 0x2200C3 = IOCTL_ADSS_IPC_CONNECTION_CREATE_UNNAMED_SERVER
  - 0x2200C7 = IOCTL_ADSS_IPC_CONNECTION_UNMARSHAL_SERVER_PORT
  - 0x2200CB = IOCTL_ADSS_IPC_CONNECTION_RELEASE_HANDLES
  - 0x2200CF = IOCTL_ADSS_IPC_CONNECTION_DISCONNECT_CONSOLE
- And these are now removed:
  - IOCTL_ADSS_ENLIGHTENED_FORK
  - IOCTL_ADSS_ENLIGHTENED_FORK_CALLBACK(_STATUS)
  - IOCTL_ADSS_FILE_SYSTEM_CALLBACK(_STATUS)

VFS FILE MARSHALLING AND INTEROP

- Using the functionality on the previous slides, as well as by adding binfmt_misc support, RS2's WSL can now launch Win32 binaries from the Linux environment
- First, Lxss can now create an 'unnamed' IPC server port and return the handle back through the COM interface
  - Linux side can connect to /dev/lxssclient which doesn't require root permissions, and unmarshal a handle to it
- Next, LxCore can marshal/unmarshal VFS inodes, for example creating the ability to marshal a Win32 process handle into a VFS inode
  - This allows IOCTLs to be sent to this inode to interact with the NT process on the other side (such as wait-for-terminate)
- Similarly, ability to do VFS marshalling allows LxCore to marshal the VFS inode corresponding to the mounted executable, so that the Windows side can unmarshal it into an actual File Object to execute
  - Bash.exe listens for messages that it receives from the binfmt handler on the Linux side – an extra instance of 'init' that connects to the unnamed server port and sends a special 'launch Win32 application' message
- Related to these changes, *inotify* support is now available over DrvFs, meaning Linux applications can watch for changes to Win32-owned files

INTEROP-RELATED IPC OVERVIEW



ref: https://blogs.msdn.microsoft.com/wsl/2016/10/19/windows-and-ubuntu-interoperability/

UNDOCUMENTED DRIVER EXPORTS

- LxCore.sys now exports all the APIs necessary to register a new “miscellaneous” family device
  - /dev/foobar of Major type MISC (10)
- Can then handle inode creation and subsequent file descriptor creation
  - Can then support read/write/seek/ioctl and all other I/O operations
- Also has full access to useful APIs to getting caller information
- But needs to know LX Instance in order to create new /dev/ entry…
  - Calling LxInitialize grants access to a “VFS Mount Notification” callback which can be used to register + insert the device entry into the instance (it’s passed through).
  - Can save Instance in context pointer used for all later operations

DEMO

3RD PARTY WINDOWS KERNEL DRIVER RESPONDING TO IOCTL FROM LINUX APPLICATION

NEW SYSCALL SURFACES

- 213 system calls (RS1) now give rise to 220 system calls (14965)
  - 7 new system calls don't paint the full picture – 100KB of new code added in 3 months
- Restartable system calls
  - Can cause interesting TOCTTOU and other timing-related issues with signal handling
- Namespaces & PID Anchors
  - Complexity of handling isolated system resources, PIDs, etc, can lead to code duplication and even more complex system call handling
- Cgroups
  - Same as above plus complexity around actually enforcing the limitations and/or interaction with Job objects
- Chroot / pivot_root
  - Same as above – complexity around isolation and handling at the file system level, complicating code paths
- These are complex issues in Linux – not specific to WSL implementations of them

VISIBILITY & SECURITY ISSUES

UNASSAILABLE FACTS ABOUT WSL

- All WSL process' handles were kernel handles
  - Handle table appears empty to all tools and software
  - Impossible to determine resource access
- No PEB, no NTDLL, no TEB, etc… and the main executable is ELF
  - Would security products ready to handle these types of processes?
- The reality is that Pico processes execute ELF binaries, so no PE files / image sections (aka no DLLs, etc.)
  - So features such as AppLocker need their teams to have resources available to add support
- Also, minor WinDbg annoyances:
  - !process does not understand Pico processes (will show "System Process" for all of them)
  - No symbols for LxCore, so cannot analyze Pico processes or their state

ATTACK SURFACE ANALYSIS

- 220 system calls that can now result in possible local privilege escalation
  - An extra 800KB attack surface!
- Linux processes, within firewall and token rules, have disk & network access like normal applications
  - Ransomelfware, anyone?
- BSODs are found – accidentally – during Insider Preview builds (all responded to & fixed by WSL team)
  - One NULL pointer dereference – fixed in RS1
  - One invalid pointer dereference (may have led to LPE) – fixed in RS1
  - Another execution of a NULL/data pointer in AFD (Kernel Winsock Shim) – fixed in RS2
  - And this is without anyone external actively fuzzing this thing!
- At least the IPC interfaces were locked down in RS1…
  - But even on RS1, an unprivileged user can replace the init daemon to exercise the IPC interfaces
- **Optional feature for now, requires admin to enable**

VISIBILITY ISSUES IN BRIEF…

- No audit / event log entry when WSL process is launched in RS1 – Fixed in RS2
- No process/thread notifications through documented API in RS1 – Fixed in RS2
  - But opt-in (vendors must update their software) – Demo: Process Monitor and WSL Binaries
- Removing PROCESS_QUERY_INFORMATION and PROCESS_VM_OPERATION | PROCESS_VM_READ makes it impossible to dump/analyze
  - Demo: Task Manager
  - Some tools can do it, but only by loading a driver and violating the policy
- No support for DWARF in Debug Engine – even with kernel assistance, can't see user-mode stacks
  - Demo: Process Hacker
- No integration with Enterprise Security: AppLocker and/or Code Integrity and/or Windows Defender, AMSI
  - As long as someone can talk to LxssManager, spawning init + other binaries bypasses all checks
  - Doable from PS, even in Constrained Language Mode ("not a security boundary")

“PROCESS” ISSUES

- WSL Project tries to do the right thing, but the road to hell is paved with good intentions, as they say…
- Open issue tracker on GitHub. People are posting BSODs and other 0-days in the driver, all in the open
  - RS2: Team has updated community guidelines to discourage this type of activity and loop in MSRC instead
  - Team actively going through bugs to remove stack traces/memory dumps/etc
- Top features are often the most dangerous to implement
  - Ping in non-admin context required allowing (limited, ICMP-only) raw sockets from Linux applications
  - Already lots of asks for /dev/mem – would expose physical memory to unprivileged applications
  - Lots of asks to expose GPGPU support for CUDA/OpenCL – increased attack surface again
- Turning on WSL requires turning on *Developer Mode* – installs SSH server/group for no related reason
  - No notification to user, and all local accounts are automatically added to remote SSH login group
  - No DoS/brute-force protection
- Actual unprivileged user->admin vulnerabilities not issued CVEs/bulletins (!)

SECURITY ISSUES

- On x86, Pico vs non-Pico interfaces aren't thoroughly protected
  - Certain interrupts/assembly sequences from Pico could result in "escape" to NT side
  - x86 is not currently supported for this product (originally was only for Windows Mobile Emulator for Astoria)
  - Could make a comeback of IoT Core/Enterprise has WSL support
- Object handle duplication is done with "KernelMode" as the Access Mode
  - Allows passing kernel handles for duplication over IPC interfaces
  - May cause BSOD if handle is marked as "protected" during close
  - Mitigated (so far) on x86 because HANDLE type in the IPC infrastructure is 32-bit, and kernel handles require sign extension
  - RS2: Recently fixed in latest builds
- Transfer of "Fork token" in IPC interface directly references TOKEN handle as ETOKEN
  - When sent to PspCreateMinimalProcess, previous mode is "KernelMode" which means SeAssignPrimaryTokenPrivilege & other checks are bypassed – less strict token checks

ADDITIONAL SECURITY ISSUES

- *init* is a special process in Linux environment – has immediate root privileges
  - Typically protected as well – even in LxCore, debugging, killing, and otherwise reading/modifying its memory is prohibited through Linux system calls
- But init is located in unprivileged, user-modifiable location on disk!
  - Launched as Medium IL to mitigate obvious issues, and interface is *untrusted* by actual LXSS stack
    - Extracted from signed LxssManager binary, but can be trivially bypassed with *oplock* attack
    - Also can be bypassed by dropping an ld.preload.so file, which again requires no privileges
  - Replacing init allows for hidden execution of WSL processes without full Ubuntu stack
    - And is unkillable/undebuggable from other Linux applications
- If user utilizes admin token to launch Bash.exe, then Bash.exe + children will execute with full Admin token, including all privileges and High IL
  - Now you have a stealthy, untouchable, user-controlled medium IL binary that has full control over Linux binaries running with high IL and a full token
  - This is true of *any* binary, not just init – but init has additional protections and stealth granted to it

LOCAL PRIVILEGE ESCALATION DUE TO ‘KERNELMODE’

- An actual practical example of issues with running everything with KernelMode as previous mode: KB3194798
  - No actual CVE or Bulletin ☹ “WSL is not a security boundary?”
- File access is done through *IoCreateFileEx*, which recognizes LxCore.sys and treats it as a kernel-mode caller
  - IO_FORCE_ACCESS_CHECK is/was not consistently used in all places
- In other cases, it’s done through *ZwCreateFile* without OBJ_FORCE_ACCESS_CHECK
- Both situations don’t use FILE_OPEN_REPARSE_POINT – which means reparse points will be followed
- This allows tricking LxCore.sys into creating/dropping files in privileged Admin-only locations by creating a symbolic link to the Windows or System32 directory, for example
  - temp\0000000000000000_ashmem is an example of a temporary file that is opened incorrectly through this path
- Reported by James Forshaw (@tyranid) and silently fixed in RS1 update (see KB above)
  - So are we not treating WSL privilege escalations as ‘real’ bugs?

CALL TO ACTION

MAKING THINGS BETTER

CALL TO ACTION: FIX VISIBILITY / MONITORING GAPS

- Allow read-only non-invasive debugging/monitoring from Win32 applications
  - Allow PROCESS_QUERY_INFORMATION and PROCESS_READ_VM rights if SeDebugPrivilege enabled
  - Provide DWARF support in WinDBG
  - AND/OR Interop between gdb/lldb and WinDBG
- Add/document additional relevant ETW events to provide audit infrastructure with WSL data
- Add public symbols for core Process(Group), Thread(Group), File Descriptor Table & VAD in LxCore
  - Add extensions to KDEXTS (or WSLEXTS) to allow seeing these processes in kernel debugger
  - Fix !process to show Pico process name (and fix the handle count bug while you're at it too…)
- Document COM Interface

CALL TO ACTION: FIX ANTIMALWARE GAPS

- Provide antimalware callbacks for:
  - ~~Pico Process & Thread Creation/Termination (with clear mapping for clone/fork/execve)~~ Fixed in RS2
  - ELF Module Loaded (Primary + additional .so files)
- Enhance WFP and FltMgr callbacks with Pico-awareness
  - Such that PreviousMode == KernelMode is less confusing when coming from LxCore
- Provide PsIsPicoProcess() and/or PsGetPicoProcessProvider() or similar to tell process owned by WSL
- Add compatibility shims/behavior to dangerous APIs such as PsGetProcessPeb() / PsGetThreadTeb() if Pico process
  - Return fake data structure if multiple AVs show NULL-pointer dereferences AND/OR
  - Add NT_ASSERT + DriverVerifier hooks to catch during development
- Consider open source / reference ELF parsing library (or Rtl APIs) to avoid buggy parsers in the kernel

CALL TO ACTION: PROCESS AROUND SECURITY

- Hire Internally, or Outsource Externally, hardcore Linux Pentesters “Red Team”
  - Fuzz the system calls
  - Fuzz the /dev devices and IOCTLs
  - Fuzz the VFS infrastructure
  - RS2: Internal fuzzing infrastructure was put into place – there will be a talk on this today
- ~~Actively monitor and lock down GitHub issues with BSOD and/or other security-sensitive details~~
  - ~~Actively discourage such issues from being posted there in the first place~~
- Fix SSH Server issues, or stop turning it on if someone enables WSL
- Put security first (SDL)
  - “I can’t ping from non-Admin” -> Make sure WSL processes can’t do arbitrary raw sockets without being Admin!
    - This was handled correctly
  - “XXX doesn’t work without /dev/mem” -> You’d allow WSL access to RAM…

CALL TO ACTION: DEFENSE-IN-DEPTH

- Use PatchGuard's "Range Protection" feature to protect the LxpRoutines array – not just system calls
  - Also evaluate if it really needs to be exported?
- Restrict kernel-mode interface inside LxCore.sys
  - Do not use export interface but rather some mechanism which can check signature (like ELAM does)
- Fix (?) DriverEntry "hijack" trick which allows arbitrary driver to take over VFS Mount Notification
- Fix "init" process handling to avoid non-authenticated users from attacking it
  - Consider locking down ld.preload.so

CALL TO ACTION: WSL SECURITY STRATEGY

- Act as thought leaders in Linux security – don't settle for "this is what Linux has", go beyond it
  - Work with Canonical on ELF Signing (three competing 'standards' exist)
  - Improve DWARF, if needed, to better support DbgHelp/DbgEng support for it (WINE does this)
- Integrate WSL binaries with Windows Defender, AppLocker and other Windows security tools
- Integrate Bash with AMSI (Anti-Malware Scanning Interface) just like PowerShell is
  - Add support for Python, Ruby, Perl, and other popular cross-platform scripting interface
- Work with clang project to add support for mitigations such as CFG, RFG, fast fail/security assertions
  - Making a new MEM_ELF_IMAGE memory manager section type would allow this + many other improvements
- Make running Linux binaries on Windows, safer than running them on Linux, not the other way around!
  - Great marketing opportunity – using Ubuntu on Windows is safer than Ubuntu on Linux

REFERENCES & GREETZ

- Sincere thanks to the following people for their assistance, support and help with this presentation and WSL-related research:
  - Arun Kishan, Nick Judge, Jamie Schwartz, Deepu Thomas, Jason Shirk, John Lento, Akila Srinivasan. Ben Hillis, Sven Groot, Simon Pope, Stephen Hufnagel, Nate Warfield, Russ Alexander
- Thanks to Ange Albertini for his amazing work on the presentation logo “Evil Penguin Hiding in Windows”
- Be sure to check out the official WSL Blog and GitHub as well as the Product Page (release notes, etc…)
  - https://blogs.msdn.microsoft.com/wsl
  - https://github.com/Microsoft/BashOnWindows
  - https://msdn.microsoft.com/en-us/commandline/wsl/
- https://github.com/ionescu007/lxss for presentation slides and code

Q & A

THANK YOU