Digital Investigation 26 (2018) S3–S11

Contents lists available at ScienceDirect

Digital Investigation

journal homepage: www.elsevier.com/locate/diin

DFRWS 2018 USA — Proceedings of the Eighteenth Annual DFRWS USA

Memory forensics and the Windows Subsystem for Linux

Nathan Lewis [c], Andrew Case [a], Aisha Ali-Gombe [d], Golden G. Richard III [b, c, *]

[a] Volatility Foundation, USA
[b] Center for Computation and Technology, Louisiana State University, USA
[c] School of Electrical Engineering & Computer Science, Louisiana State University, USA
[d] Department of Computer and Information Sciences, Towson University, USA

Keywords:
Memory forensics
Computer forensics
Memory analysis
Windows 10
Linux
WSL

A B S T R A C T

The Windows Subsystem for Linux (WSL) was first included in the Anniversary Update of Microsoft's Windows 10 operating system and supports execution of native Linux applications within the host operating system. This integrated support of Linux executables in a Windows environment presents challenges to existing memory forensics frameworks, such as Volatility, that are designed to only support one operating system type per analysis task (e.g., execution of a single framework plugin). WSL breaks this analysis model as Linux forensic artifacts, such as ELF executables, are active in a sample of physical memory from a system running Windows. Furthermore, WSL integrates Linux-specific data structures into existing Windows data structures, such as those used to track per-process metadata as well as userland runtime data. This integration results in existing analysis plugins producing inconsistent results when analyzing native Windows processes compared to WSL processes. Further complicating this situation is the fact that much of the WSL subsystem internals are completely undocumented. To remedy the current deficiencies related to WSL analysis, a research effort was undertaken to understand which existing Volatility plugins are affected by the introduction of WSL as well as what updates are necessary to fully support memory forensics of WSL. This paper describes these efforts, including our study of the operating systems data structures relevant to WSL as well as the development of new Volatility analysis plugins.

1. Introduction

The Windows Subsystem for Linux (WSL) (The Windows Subsystem for Linux, 2017) is a significant new feature that was introduced in the Anniversary Update of Microsoft's Windows 10 operating system. WSL provides the first truly native support for Linux applications on a Windows operating system by implementing loading and execution of ELF applications and libraries. The ability to run native ELF files brings a large and diverse set of existing Linux applications to Windows users, such as web, email, FTP, and SSH servers, as well as a full suite of end-user applications. Along with providing a simple method for transitioning existing applications from Linux to Windows, Microsoft has also pledged a long-term commitment to WSL as reflected in its documentation (MSDN, 2017) and in the large set of updates and new features that were included in the Fall Creators Update (Raj, 2017). The combined effect of these actions suggests that WSL will be present and supported for many years and that defensive security practices must account for its existence.

Unfortunately, the introduction of a new executable file format into Microsoft Windows, along with a very large number of new Linux applications, provides an immense challenge for endpoint software security vendors, such as anti-virus companies (Ionescu, 2016a). While these companies have dedicated nearly two decades of research to understanding and detecting threats from Portable Executable (PE) format files, the native Windows executable file format, the very recent introduction of ELF requires an entirely new set of detection capabilities and algorithms. As described in Section 3, not only does the file new format provide challenges, but the architecture that supports ELF files also introduces many new data structures that make traditional malware detection techniques inadequate.

This gap in traditional Windows analysis techniques affects not only runtime software security vendors, but also memory forensics frameworks, since these frameworks are very sensitive to the location and layout of data structures populated by the operating system. Specifically, the ability to correctly locate and parse these

* Corresponding author.
E-mail addresses: nplewis@lsu.edu (N. Lewis), andrew@dfir.org (A. Case), aaligombe@towson.edu (A. Ali-Gombe), golden@cct.lsu.edu (G.G. Richard).

https://doi.org/10.1016/j.diin.2018.04.018

data structures is a fundamental design component of memory forensics tools and similarly, the ability to locate all relevant memory-resident artifacts is a requirement for thorough malware and anomaly detection. The introduction of new data structures and algorithms by WSL breaks many existing algorithms implemented by current analysis frameworks. Furthermore, a class of malware known as *bashware* can programatically enable WSL and execute malicious code while taking advantage of the obfuscation provided by WSL (Elbaz and Atias, 2017).

To close the detection gaps currently available to attackers through WSL, we conducted research to document the new sources of forensics artifacts produced by WSL as well as creating new memory forensics algorithms that provide better coverage of the WSL subsystem. This paper describes this research and its outcomes, including discussion of the relevant WSL architectural components, the deficiencies in existing memory forensic algorithms, and the new algorithms we created to recover WSL-related memory artifacts. Our research was conducted through reverse engineering of the WSL userland and kernel components as well as testing and creation of Volatility (The Volatility Framework, 2017) plugins. Volatility was chosen as our target memory analysis framework because of its widespread use throughout the digital forensics community combined with its ample documentation. All of our newly created Volatility plugins, along with our patches to existing plugins, will be contributed to the upstream project upon publication of this paper.

2. Related work

2.1. WSL architecture memory analysis research

Internal components of the WSL architecture are closed source and sparsely documented by Microsoft. While Microsoft's MSDN and Windows Internals 7th Edition (Yosifovich et al., 2017) document the high-level design ideas and exported APIs, these references do not describe data structures or algorithms utilized by WSL. Microsoft also does not provide full Visual Studio debugging files (generally referred to as PDB files) for the WSL subsystem.

The only substantial existing memory analysis research for WSL was undertaken by Alex Ionescu and appeared in Blackhat 2016 (Ionescu, 2016a). Code, in the form of WinDbg scripts, related to this effort is publicly available in a Github repository (Ionescu, 2016b). A complete comparison between our research effort and his is provided in Section 4.

Concurrently with our research effort, a member of the Volatility development team, Michael Ligh, published a set of patches that enabled correct reporting of WSL process names (Ligh, 2017). Our team had performed the same research, as discussed in Section 5.

2.1.1. Cygwin for Linux on Windows

Executing Linux programs on Windows systems was possible before the release of WSL. Cygwin is a software project that allows users to execute Linux programs in Windows environments. The Cygwin terminal provides a shell environment from which users can interact with a virtual filesystem, execute supported programs, and issue POSIX system calls (Cygwin, 2017). The Cygwin design is similar to WSL in that both bring lightweight virtualization of Linux environments to Windows systems. However, the ways in which this functionality is provided are significantly different. Cygwin compiles Linux source code into standard PE-formatted executables, which are then linked against a library that provides POSIX compatibility by translating between Unix and Windows system calls. Notably, Cygwin does not introduce ELF files into Windows and operates entirely in userspace, without kernel components. In contrast, WSL is more tightly integrated, introduces support for executing ELF files, and has both userland and kernel space components.

3. WSL background

Microsoft's Drawbridge project team focused its research efforts on application sandboxing, a method for lightweight virtualization. The project's goal was to introduce a library operating system model into a commercial version of Windows that relocated operating system dependencies of sandboxed applications into their process' address spaces (Baumann et al., 2016). Drawbridge first produced a prototype version of Windows 7 using a library OS architecture in 2011 (Porter et al., 2011).

Drawbridge proposed two new process types - *minimal* and *pico* - while retaining support for Microsoft's traditional NT processes. Unlike NT processes, minimal processes lack key Window components that tie NT processes directly to the kernel. Fig. 1 depicts these components. Minimal processes have empty userland memory and are unmanaged by the kernel in many respects. Pico processes are minimal processes that are also associated with a corresponding kernel driver. A pico process' kernel driver is responsible for managing the process' userland memory, threads, scheduling, file handles, and sockets (Hammons, 2016a; Hron, 2017). This driver is commonly referred to as the *pico provider*.

WSL, the most prominent application of pico processes in Windows, was released in 2017 with the 64-bit version of the Windows 10 Fall Creators Update after more than one year of beta testing (Turner, 2017). It enables users to directly execute userland Linux programs in Windows 10 by associating each executing Linux application with a pico process. This allows users to execute ELF binaries without the need for a virtual machine, source code modification, or an intermediate application. Furthermore, users can download an app for each of the five currently supported Linux distributions from the Microsoft Store (Cooley et al., 2017): Ubuntu, Debian GNU/Linux, openSUSE Leap 42, SUSE Linux Enterprise Server 12, and Kali Linux. The following processes are components of WSL's implementation and are illustrated in Fig. 2:

- *wsl.exe* or *bash.exe*: A userland command line process through which users interact with WSL. This program can be instantiated more than once.
- *LxssManager*: A Windows service that facilitates communication between *wsl.exe*/*bash.exe* processes and the WSL pico provider.
- *lxss*: A Windows system service that serves as the WSL pico provider.
- */init*: A Linux pico process that facilitates communication between Windows processes and its descendants. *lxss* creates one */init* process per instantiated Linux distribution.
- */bin/bash*: A Linux pico process that supports the WSL shell program. Each *wsl.exe* and *bash.exe* process is paired with a matching */bin/bash* process.

To start WSL, a user executes the *<distro>.exe* program corresponding to a desired Linux distribution, which creates a *wsl.exe* process. A user can also access the system's default distribution by executing *bash.exe* or *wsl.exe* directly. Each execution is isolated by Windows in its own *Linux instance*. The WSL NT services and an/*init* pico process will be created for the user's Linux instance if they don't already exist. The *lxss* service registers itself as the pico provider with the Windows kernel through the `PsRegisterPicoProvider` system call. This instructs the kernel to allow lxss to manage system calls, exceptions, and resources on behalf of WSL pico processes (Hammons, 2016a). A Linux shell GUI will be created if wsl.exe is executed either from within cmd.exe or from the Windows GUI.