Behavior Analysis and Detection of Bashware

Andrei Mermeze
email andrei.mermeze@gmail.com

Advisor: Phd. Dragos Radu

Contents

1. Introduction

1.1 Motivation

With the release of Windows Subsystem for Linux (WSL). Consisting of two new kernel-mode drivers, lxcore.sys and lxss.sys, that implement more than 100 linux system calls, it opened an attack surface that was not covered by monitoring tools or AV vendors.

1.2 Objectives

The main objective is to provide a partial solution to this new security problem. It should defend a system from bashware, suspicious interraction between WSL processes and Windows processes and to provide system administrators with enough logs to properly respond to incidents, while not disrupting the user experience.

1.3 Personal Contributions

1.4 Thesis Outline

2. State of the art

2.1 Windows Subsystem for Linux

2.1.1 WSL Main Components

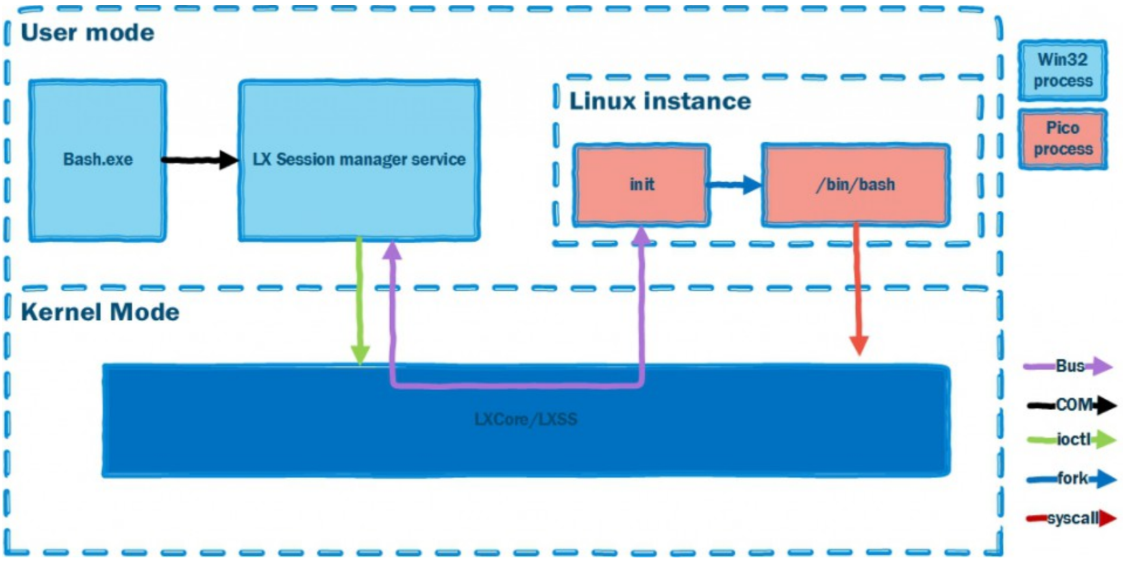

Figure 2.1: WSL Compoents [4]

2.1.2 WSL File System

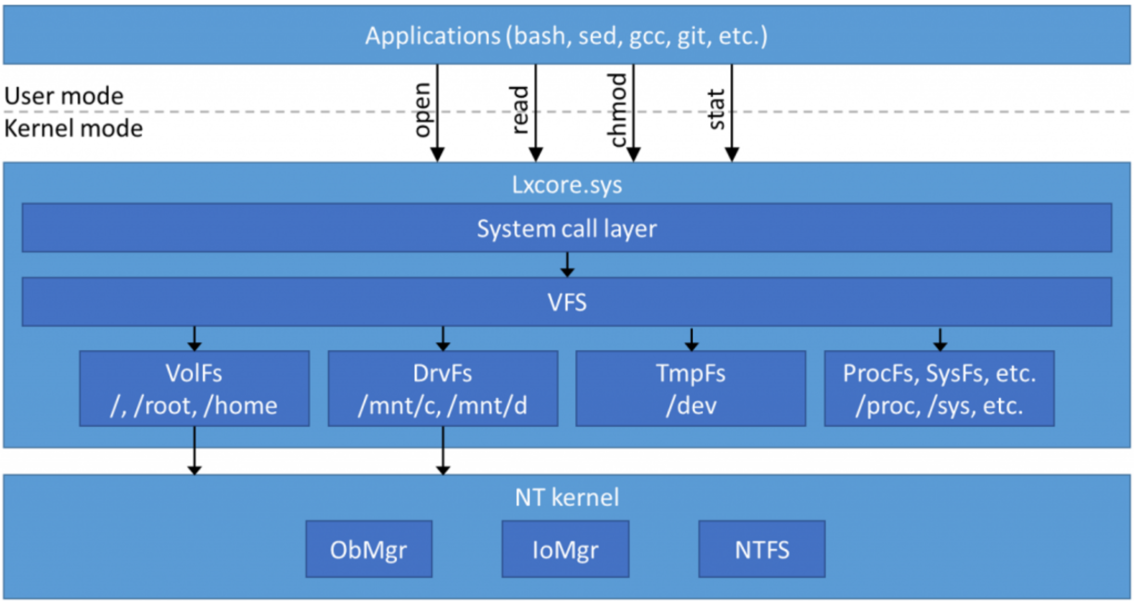

Figure 2.2: WSL File System [5]

2.2 Linux Malware

2.2.1 Process Injection

2.2.2 Process Hollowing

2.3 Bashware and Exploits

2.3.1 execve exploit

2.3.2 File Infector

2.4 Behavioral Detection

3. Used technologies

3.1 WDM

3.1.1 Legacy Filter Drivers

3.1.2 File System Legacy Filter and Minifilter Drivers

A Legacy Filter driver is a kernel-mode that could attach to a device's stack. In the context of file system filtering, these filter drivers could intercept file system I/O operations. Developing legacy filter drivers was quite troublesome and led to many incompatibilities between filter drivers. This is one of the reasons for which minifilter drivers were added.

Minifilters have the same abilities as file system legacy filter drivers, but they are easier to develop and are overall safer. Their load order no longer depends on the attach order, but on a predefined value named altitude. Minifilters are managed by FltMgr, which is a legacy filter driver implemented by microsoft.

3.2 C++ in Kernel Drivers

By default, msvc does not support C++ for kernel-mode drivers.

3.3 .NET Framework

3.4 WCF

3.5 UWP

4. Architecture

4.1 High Level Overview

The system is composed of 2 main components

- wslmon - an sdk that contains the detection logic as well as communication logic between kernel-mode and user-mode
- wslam - the integrator

4.1.1 wslmon

Wslmon is a software development kit that could be integrated by any OEM to include it in an existing security solution. It consists of:

- wslflt.sys - a minifilter driver that encapsulates the monitoring, analysis and detection logic
- wslcore.dll - a shared library that provides an interface to communicate with the minifilter driver

wslflt.sys is C++ minifilter driver that contains the requirued sensors for monitoring process' activity. It's key components are the process filter, file filter, event dispatcher, the communication framework and the detection heuristics.

4.1.2 wslam

Integrates the wslmon SDK. It consists of:

- uwpcomm.dll - WinRT
- wslamsscomcore.dll - C++/CLI
- wslamss.exe - .NET Service
- WSL Anti-Malware.exe - UWP Store application
- NotificationsListener - Background app service deployed by the WSL Ant-Malware.exe store app

uwpcomm.dll uses WinRT in order to send notifications to the background app service via an AppServiceConnection.

wslammsscore.dll is built on top of C++/CLI (Common Language Infrastracture) to wrap uwpcomm.dll and wslcore.dll and export managed .NET classes for use in the .NET service.

wslamss.exe contains the integration logic (ie. exceptions, notifications) and servers as a communication endpoint, built using WCF framework, for the store application.

WSL Anti-Malware.exe is a windows store app that

NotificationsListener is an app service, implemented as a background task, that listens to integrator notifications even if the store application is closed, and notifies the user about specific events via toast notifications

5. Implementation Details

5.1 Process Filtering

```
PsSetCreateProcessNotifyRoutineEx2(a, b);
```

5.2 File Filtering

5.3 Communication

5.4 Detection Logic

5.5 Scoring Engine

6. Testing

Rigorous testing, especially in the context of kernel-mode modules and security solutions, is essential in order to provide a stable, usable and efficient product. Combining whitebox and blackbox techniques together with test driven developement paradigms throughout the development of a product is crucial in order to offer a high quality product.

6.1 Whitebox testing framework

In order to use the Test Driven Development paradigm, a whitebox unittesting framework was required. The unittest project contains mock definitions of kernel functions (e.g. ExAllocatePoolWithTag), and unittests for each class. Mocking kernel functions helped in achieving high coverage of the driver code in a user-mode enviroment, thus saving development time. Moreover, these unittests can be easily configured to run at each solution build, making it a very reliable and easy to use continuous integration tool.

The other alternative was having another kernel-mode driver that would test the API exported by wslflt.sys. This is an unreliable and slow testing method because most bugs would cause a blue screen and can even corrupt the testing enviroment. Moreover, analysing kernel dumps in order to find bugs is a much slower process compared to analysing user-mode crashes or exceptions. Also, building a continuous integration system around this testing framework is a lot more complicated because it involves virtual machines, automatiically applying OS images and re-applying them in case a BSOD occurs.

6.2 Blackbox testing

Blackbox testing, both manual and automated, is the main method of attesting a product's value, usefuleness and correctness regarding the users' needs.

6.3 Performance testing

6.4 WHQL Testing

Windows Hardware Quality Lab[3] tests involve running various testcases regarding Disk I/O, code integrity checks, pipes I/O, transactional I/O and more.

In order to release a driver to the market, it needs to be digitally signed by microsoft, and the only way of receiving the signature, it must pass the WHQL test suite. In the case of wslflt.sys, that would be the Filter Driver Test Suite[2].

7. Heuristics and Detection Algorithms

7.1 Privilege Escalation Detection

7.2 File Infector Detection

7.3 Ransomware Detection

8. Performance impact analysis

9. Possible improvements

9.1 Network Filtering

A network filter kernel-mode driver is required in order to monitor and report suspicious network activity inside WSL. For example, such a component could possibly detect reverse shell behavior.

9.2 User-Mode Hooking Framework

Injecting a shared library into monitored process could be (for now) the only way to detect WSL process hollowing or process injection.

References

[1] https://docs.microsoft.com/en-us/windows-hardware/drivers/kernel/introduction-to-wdm/.

[2] https://docs.microsoft.com/en-us/windows-hardware/test/hlk/testref/filter-driver-tests/.

[3] https://docs.microsoft.com/en-us/windows-hardware/test/hlk/windows-hardware-lab-kit/.

[4] https://blogs.msdn.microsoft.com/wsl/2016/04/22/windows-subsystem-for-linux-overview/.

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

[6] Saar Amar. *execve exploit*. https://github.com/charlespwd/project-title.

[7] Saar Amar. "LINUX VULNERABILITIES, WINDOWS EXPLOITS Escalating Privileges with WSL". Bluehat IL. 2018.

[8] Manuel Egele, Theodoor Scholte, Engin Kirda, and Christopher Kruegel. "A Survey on Automated Dynamic Malware Analysis Techniques and Tools". In: *ACM Computing Surveys (CSUR)* 44.6 (Feb. 2012).

[9] Alex Ionescu. "GAINING VISIBILITY INTO LINUX BINARIES ON WINDOWS: DEFEND AND UNDERSTAND WSL". Bluehat. 2016.

[10] Alex Ionescu. "THE LINUX KERNEL HIDDEN INSIDE WINDOWS 10". Blackhat. 2016.

[11] Cem Kaner and Rebecca L. Fiedler. *Foundations of Software Testing*. 2013.

[12] Michael Hale Ligh, Andrew Case, Jamie Levy, and Aaron Walters. *The Art of Memory Forensics*. John Wiley & Sons, Inc., 2014.

[13] Pavel Yosifovich, Alex Ionescu, Mark E. Russinovich, and David A. Solomon. *Windows Internals, Part 1: System architecture, processes, threads, memory management, and more*. 2017.