Behavior Analysis and Detection of Bashware

Andrei Mermeze
email andrei.mermeze@gmail.com

Advisor: Phd. Dragos Radu

Contents

1 State of the art **2**
1.1 Windows Subsystem for Linux 2
1.1.1 WSL Main Components 2
1.1.2 WSL File System 3
1.2 Linux Malware 3
1.2.1 Process Injection 3
1.2.2 Process Hollowing 3
1.3 Bashware and Exploits 3
1.3.1 execve exploit 3
1.3.2 File Infector 3
1.4 Behavioral Detection 3

2 Used technologies **4**
2.1 WDM 4
2.1.1 Legacy Filter Drivers 4
2.1.2 File System Legacy Filter and Minifilter Drivers 4
2.2 C++ in Kernel Drivers 4

3 Architecture **5**
3.1 High Level Overview 5
3.2 wslflt.sys 5
3.2.1 File Filter 5
3.2.2 Process Filter 5
3.2.3 Event Dispatcher 5
3.2.4 Communication Framework 5
3.2.5 Heurisitcs 5
3.3 wslcore.dll 6
3.4 wslsvc.exe 6
3.5 wslam.exe 6

4 Implementation **7**
4.1 Process Filter Implementation 7

5 Testing **8**
5.1 Whitebox testing framework 8
5.2 Blackbox testing 8

5.3 Performance testing 8
5.4 WHQL Testing 9

6 Heuristics and Detection Algorithms 10

7 Performance impact analysis 11

8 Possible improvements 12
8.1 Network Filtering 12
8.2 User-Mode Hooking Framework 12

References 13

1. State of the art

1.1 Windows Subsystem for Linux

1.1.1 WSL Main Components

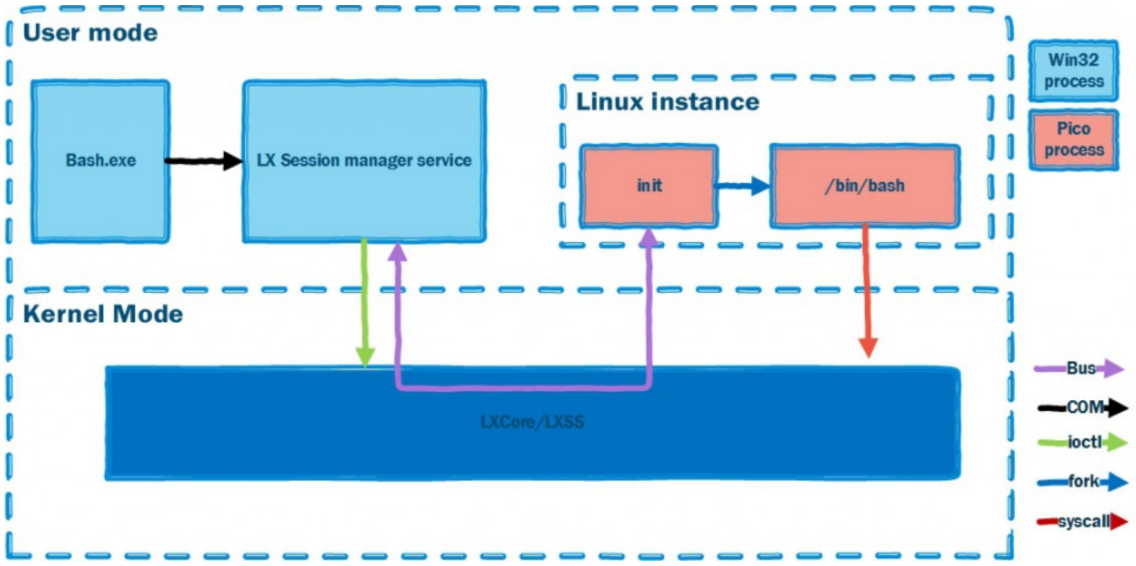

Figure 1.1: WSL Compoents [4]

1.1.2 WSL File System

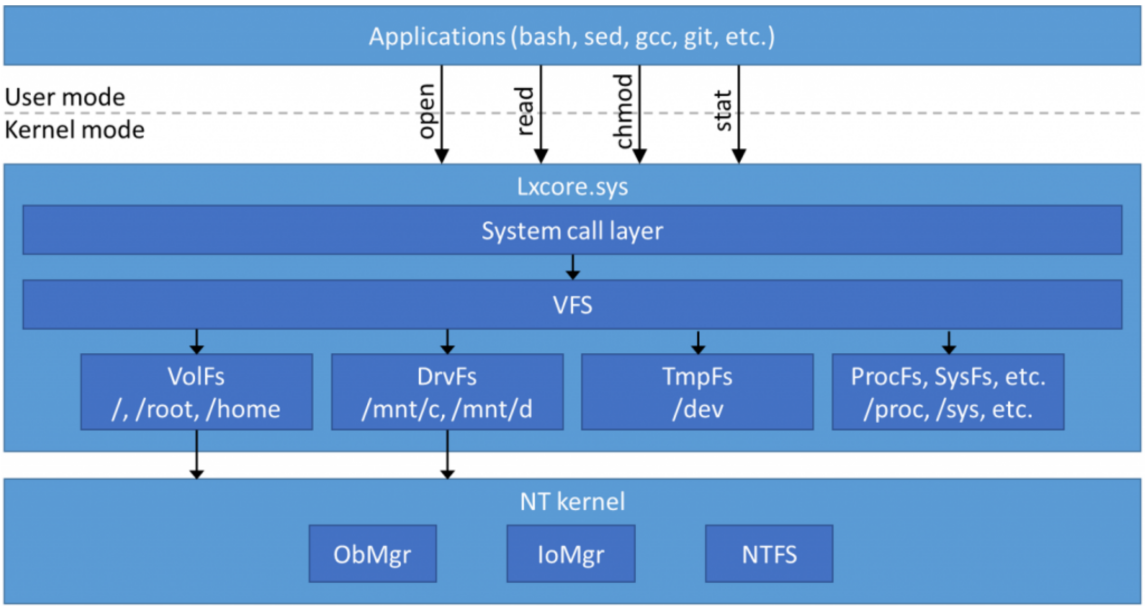

Figure 1.2: WSL File System [5]

1.2 Linux Malware

1.2.1 Process Injection

1.2.2 Process Hollowing

1.3 Bashware and Exploits

1.3.1 execve exploit

1.3.2 File Infector

1.4 Behavioral Detection

2. Used technologies

2.1 WDM

2.1.1 Legacy Filter Drivers

2.1.2 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.

2.2 C++ in Kernel Drivers

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

3. Architecture

3.1 High Level Overview

The system is composed of 4 main components

- wslflt.sys
- wslcore.dll
- wslsvc.exe
- wslam.exe

3.2 wslflt.sys

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.

3.2.1 File Filter

The file filter's role is to filter disk I/O.

3.2.2 Process Filter

The process filter is notified of process and thread creation and termination by the windows kernel.

3.2.3 Event Dispatcher

3.2.4 Communication Framework

3.2.5 Heurisitcs

3.3 wslcore.dll

This dll is an abstraction of the wslflt.sys driver. It contains the communication logic between kernel-mode and user-mode and exports multiple callbacks to an integrator. It's main purpose is to hide the filtering and detection logic and to provide an easy way to integrate the system in a complete security solution.

3.4 wslsvc.exe

Wslsvc.exe is Windows service that integrates the previously mentioned DLL.

3.5 wslam.exe

Wslam.exe is the GUI component of the system. It provides the user with logs about processes' actions and notifies him about detections or other suspicios activity.

It is an UWP .NET application that communicates with wslsvc via TCP/IP

4. Implementation

4.1 Process Filter Implementation

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

5. 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 black-box techniques together with test driven developement paradigms throughout the development of a product is crucial in order to offer a high quality product.

5.1 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. ExAllocatePool-WithTag), and unittests for each class. Mocking kernel functions helped in achieving high coverage of the kernel-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.

5.2 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.

5.3 Performance testing

5.4 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].

6. Heuristics and Detection Algorithms

7. Performance impact analysis

8. Possible improvements

8.1 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.

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

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

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