Looking for Linux: WSL Key Evidence

Amanda Draeger
Abstract

Microsoft released Windows Subsystem for Linux (WSL) in 2016 to much fanfare, but little research into the security implications of installing this feature followed. This lack of research, and lack of documentation, is a problem for the administrators who want to take advantage of its feature set while monitoring their systems for unusual behavior. Native Windows logging can provide visibility into WSL’s behavior, but there has been no research on which logs can provide this visibility, and what exact information they can provide. This paper examines how to monitor a Windows 10 system with WSL installed for common indicators of malicious activity.
1. Introduction

Microsoft introduced Windows Subsystem for Linux (WSL) at Build 2016 (Kirkland, 2016). While the technology world expressed great interest in WSL, the security community has not published much research since its release. This lack of knowledge leaves defenders at a disadvantage, as there are currently no credible references for them to use when conducting incident response on a system that has WSL installed. The goal of this research is to identify forensic artifacts that defenders can use to identify and track potentially malicious activity.

As of the time of writing, there were no reports of real-world attackers using WSL as part of their attack chain. However, CheckPoint researchers theorized an attack vector that they dubbed Bashware (Khandelwal, 2017). The Bashware technique exploits the WSL architecture to hide from most Windows security software. To understand how this may be possible, one should first understand what Microsoft did to allow native Linux executables to run on Windows.

WSL Version 1 does not use virtualization or containerization to run Linux executables (Kirkland, 2016). Instead, WSL is roughly analogous to emulation, not unlike the emulation that wine performs to allow Windows executables to run on Linux systems. Microsoft took the Linux user space (initially only Ubuntu, but they have since expanded the number of distributions) and translated those syscalls into Windows syscalls. To perform that translation, WSL uses pico processes and pico providers.

Pico processes do not look like standard Windows processes in that they don't have most of the information generally assigned to processes, like a Win32 Process Environment Block (Hammons, 2016). Pico processes rely on a pico provider to translate the Linux process into a Windows process. The pico provider acts as a Windows kernel-mode driver, processing syscalls from user space.

WSL uses two pico providers: lxss.sys and lxcore.sys (Hammons, 2016). These two Windows drivers translate between Linux processes and the Windows NT kernel. The pico processes are the programs that the user interacts with from inside the WSL.

Amanda Draeger, amanda@tindrasgrove.com
environment. Two examples of pico processes are `init` and `/bin/bash`. Figure 1 illustrates the relationship between pico processes, pico providers, and Win32 processes.

![WSL components diagram](image)

*Figure 1. WSL components (Hammons, 2016).*

2. Research Method

The lab environment for this research used two virtual machines: one with WSL installed, and one without WSL installed. This setup allowed the behavior between the two configurations to be compared. To ensure the two systems were configured as similarly as possible to each other, the system without WSL was cloned, then WSL was installed on the cloned system. During the research process, many audit and logging changes were made to the systems; these changes will be discussed when addressing the data gained from these sources.

The general testing methodology was to replicate one of the activities listed in Section 2.2. If that activity left log data, it was recorded. Otherwise, auditing and logging configurations were changed until that activity left log data that could be used during incident response.

Amanda Draeger, amanda@tindrasgrove.com
2.1. Tools

Microsoft supplies virtual machines to the public for testing and development purposes (Microsoft Corporation, 2019). This research used one of these freely available virtual machines. This system ran Windows 10 Enterprise Evaluation, version 1809, build 17763.737, and PowerShell version 5. Windows 10 was at the most recent patch level at the time of research. There are several choices of Linux available in the Microsoft store for WSL. Ubuntu was used because it was the original version available, and because it is one of the most commonly used.

This research was conducted without using any third-party security products. The goal of this was to understand how WSL behaves and communicates with the underlying Windows operating system. The focus was on using Windows-native logging, including Sysmon, to observe the activity. The result is research that is entirely vendor-agnostic and that increases understanding of how these processes communicate.

2.2. Activity

When deciding what activity would be the focus of this research, the primary reference was MITRE’s Adversarial Tactics, Techniques, and Common Knowledge (ATT&CK) (MITRE, 2019). ATT&CK is a “knowledge base of adversary tactics and techniques based on real-world observations” (MITRE, 2019). This document provides a basis for what activities defenders should be looking for within their environments. ATT&CK divides into tactics (broad categories) and techniques (which fall within those tactics).

Looking for every technique within ATT&CK would both be not practical for the scope of this research, and also not possible, as some of the activities are system-specific or network-centric. This research focuses on seven activities which are inherent in many of the techniques. Defenders must record these activities to understand the full scope of malicious behavior.

One of the ATT&CK tactics is Execution (The MITRE Corporation, 2019). The 33 techniques within this tactic relate to code execution on a host. The techniques include execution resulting from human interaction, automated scripts, and scheduled tasks.
Records of process execution are a precious tool for incident responders who work to identify the root cause of the behavior. WSL can make understanding process execution difficult as Linux executables are being run. Additionally, the user may execute Windows processes from within the Linux environment, and Linux processes from PowerShell. In addition to seeing what processes successfully executed, getting a complete history of the commands entered at the Linux command prompt can prove useful.

Monitoring and logging process execution at runtime is relatively straightforward. How do processes running as daemons differ from those manually executed? Is it possible to differentiate a daemonized process from a standard process? Another thing to examine is how to identify which processes have network connections, and what process initiated a particular network connection.

Another tactic is Collection (The MITRE Corporation, 2019). The techniques within this tactic deal with the gathering of information. In many cases, this means accessing files. To detect this activity, auditing access to high-value data is necessary. This research included examining how to audit WSL access to Windows files, and Windows access to WSL files.

3. Findings and Discussion

3.1. Analysis

By default, Windows does not have much log collection enabled. Gaining visibility into the selected activities required changing logging and auditing configurations. These changes, as well as the resulting logs and what can be learned from them, are presented in the following section. Log files are truncated for brevity.

3.1.1. Windows Process Execution

When initially released, PowerShell did not have very much logging capability. PowerShell version 5 significantly increased the visibility into PowerShell activity. This increase includes module logging, script block logging, and transcripts (Dunwoody, 2016). Administrators can add command-line process auditing to gain even more

Amanda Draeger, amanda@tindrasgrove.com
visibility into exactly how processes are executing (Turner, 2017). The configuration in Figure 2 enables module logging, script block logging, and PowerShell transcription.

![Figure 2. Local security policy configuration for process logging.](image)

For research purposes, module logging was set to log all modules (*), while all other settings were left at their enabled defaults. Since it lists every module that is loaded with every command run, module logging may not be advisable in all environments. The logs for module and script block logging are in the Microsoft Windows Powershell/Operational log. Event 4104 (Execute a Remote Command) had the most information regarding PowerShell commands. The log data below is truncated.

```xml
<Provider Name="Microsoft-Windows-Powershell" Guid="{a0c1853b-5c40-4b15-8766-3cf1c58f985a}" />  
<EventID>4104</EventID>  
<Execution ProcessID="4800" ThreadID="4220" />  
<Channel>Microsoft-Windows-Powershell/Operational</Channel>  
<Data Name="ScriptBlockText">ping tindrasgrove.com</Data>  
<Data Name="ScriptBlockId">a3cb64a8-f50c-4be2-b737-c26e72e37a17</Data>
```

These logs do not contain the data that was output to the user. For that, one must look at the PowerShell transcripts. By default, the transcript is a text file placed in C:\Users\<UserName>\Documents\<Date>. Within the folder for each day, the transcript gets separated into different files based on the source. For example, working from a PowerShell window is one source, while running PowerShell commands from a cmd.exe window is a different source, while using the PowerShell ISE is yet another source. For the incident responder, this means having to piece together data from multiple files in order to gain a single timeline. Below is an example of a PowerShell transcript logging output of a script.

```
********************
```

Amanda Draeger, amanda@tindrasgrove.com
Command start time: 20190921222318
*****************************
PS C:\Users\IEUser\Desktop\Scripts> .\LocalAccounts.ps1

PSComputerName : MSEDGEWIN10
Name            : Administrator
Status          : Degraded
Disabled        : True
AccountType     : 512
Lockout         : False
PasswordRequired: True
PasswordChangeable: True
SID             : S-1-5-21-3461203602-4096304019-2269080069-500

The transcript logs the commands run and the output of the script, but not the contents of the script. Script block logging, however, does provide the contents of the script. This activity is also in the PowerShell/Operational log, Event 4104. However, to get a complete picture, the analyst must look at multiple events. The first is an event logging that a script ran.

![Figure 3. Log showing script execution.](image)

This is followed by an event logging the contents of the script, as well as the location of the script as displayed by Figure 4.

![Figure 4. Log showing contents of script that executed.](image)

Amanda Draeger, amanda@tindrasgrove.com
The above configuration allows administrators to log commands executed from PowerShell or PowerShell commands executed from other sources. However, there are many different ways that processes can run on Windows. To gain full visibility of everything happening on a system, an analyst must log process creation. Turner (2017) shows how to set this up. For this research, both *Audit Process Creation* and *Include command line in process creation events* were enabled. These two settings, when combined, log the creation of all processes and the effective command used to execute those processes to the Security log, Event 4688. For example, running the command `ping www.tindrasgrove.com` generated the log below (truncated). Note how the process command line includes the full path to ping.exe, even though that is not the command entered. This means the analyst must rely on some of the PowerShell log types to identify what commands the user ran.

```
Process Information:
New Process ID: 0x10a4
New Process Name: C:\Windows\System32\PING.EXE
Token Elevation Type: %%1938
Mandatory Label: Mandatory Label\Medium
Creator Process ID: 0x1b4c
Creator Process Name: C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
Process Command Line: "C:\Windows\system32\PING.EXE"
```

An important caveat with logging process creation is that PowerShell cmdlets do not spawn distinct processes. Process creation was logged with `ping.exe` because it is not a PowerShell cmdlet. The equivalent cmdlet, *Test-NetConnection*, does not log a created process, although it is in the PowerShell logs. The exception to this is that running PowerShell commands from outside of PowerShell (e.g., from cmd.exe) will log a created process because PowerShell is a newly created process in this case.

### 3.1.2. WSL Process Execution

With the above changes made to Windows logging, one can now observe WSL’s behavior regarding process execution. Approximately three dozen process creation events were logged upon launching the Ubuntu executable. There are two items of particular

Amanda Draeger, amanda@tindrasgrove.com
Looking for Linux: WSL Key Evidence

Amanda Draeger, amanda@tindrasgrove.com

interest in the log below (truncated). The first is that the process is created by IEUser, who is the Windows user logged in. This contrasts with the account created for use within Ubuntu, which is named tindra. The other item of interest is the process name. This gives the path within the Windows file structure for the Ubuntu file system.

Creator Subject: MSEDGEWIN10\IEUser
Account Name: IEUser
Account Domain: MSEDGEWIN10
Process Information:
New Process ID: 0x18a8
New Process Name: C:\Users\IEUser\AppData\Local\Packages\CanonicalGroupLimited.UbuntuonWindows_79rhkp1fndgsc\LocalState\rootfs\bin\bash
Creator Process ID: 0x1d18
Creator Process Name: C:\Users\IEUser\AppData\Local\Packages\CanonicalGroupLimited.UbuntuonWindows_79rhkp1fndgsc\LocalState\rootfs\bin\bash
Process Command Line:
-bash

Sysmon can provide a slightly different view of process creation. After installing the service, Sysmon creates its event logs in Sysmon/Operational. Event ID 1 is Process Creation. One advantage of Sysmon logs is that launching Ubuntu only generated three events. Another advantage is that it logs the Process ID in decimal, while the Security log uses hexadecimal. Excerpts from the Sysmon process creation logs are below.

Process Create:
ProcessId: 5952
Image: C:\Program Files\WindowsApps\CanonicalGroupLimited.UbuntuonWindows_1804.2019.521.0_x64__79rhkp1fndgsc\ubuntu.exe
CommandLine: "C:\Program Files\WindowsApps\CanonicalGroupLimited.UbuntuonWindows_1804.2019.521.0_x64__79rhkp1fndgsc\ubuntu.exe"
CurrentDirectory: C:\Windows\system32\User: MSEDGEWIN10\IEUser
ParentProcessId: 5008
ParentImage: C:\Windows\explorer.exe
ParentCommandLine: C:\Windows\Explorer.EXE

Process Create:
ProcessId: 972
Image: C:\Windows\System32\wsl.exe
FileVersion: 10.0.17763.615 (WinBuild.160101.0800)
Description: Microsoft Windows Subsystem for Linux Launcher
Product: Microsoft® Windows® Operating System
Company: Microsoft Corporation
OriginalFileName: wsl.exe
CommandLine: C:\Windows\system32\wsl.exe ~ -d Ubuntu
User: MSEDGEWIN10\IEUser
ParentProcessId: 5952
ParentImage: C:\ProgramFiles\WindowsApps\CanonicalGroupLimited.UbuntuonWindows_1804.2019.521.0_x64__79rhkp1fndgsc\ubuntu.exe
ParentCommandLine: "C:\ProgramFiles\WindowsApps\CanonicalGroupLimited.UbuntuonWindows_1804.2019.521.0_x64__79rhkp1fndgsc\ubuntu.exe"

Process Create:
ProcessId: 6792
Image: C:\Windows\System32\lxss\wslhost.exe
FileVersion: 10.0.17763.615 (WinBuild.160101.0800)
Description: Microsoft Windows Subsystem for Linux Background Host
Product: Microsoft® Windows® Operating System
Company: Microsoft Corporation
OriginalFileName: wslhost.exe
CommandLine: C:\Windows\System32\lxss\wslhost.exe {36d5b040-584d-4fa6-b207-ale15267c78e} 456 464 468
User: MSEDGEWIN10\IEUser
ParentImage: C:\Windows\System32\wsl.exe
ParentCommandLine: C:\Windows\system32\wsl.exe ~ -d Ubuntu

Sysmon also records the logged user as IEUser, the Windows user. The directory listed does not lead to the Ubuntu file system, which is an advantage of the Security logs. However, wsl.exe and wslhost.exe (which are not listed in the Security logs) are good executables to look for if one is trying to see if WSL is running on systems. Each method of logging process creation provides unique information.

Processes executed from within the Ubuntu environment leave different traces from the launch of the Ubuntu environment itself. The command to test this activity was ping -c 4 tindrasgrove.com. This command terminates the ping after sending four packets. Interestingly, Sysmon's process creation logs did not log this event. However, Sysmon’s process termination (Event ID 5) did log the termination of the ping process.

Amanda Draeger, amanda@tindrasgrove.com
This amount of logging is less than ideal. Further, several commands (ls, cat, vi) generated neither process creation nor process termination logs. Fortunately, the Security process creation logs (Event ID 4688) fully captured the process, as well as the command’s arguments.

One command that the Windows process creation log did not log was history. As with PowerShell cmdlets, history does not meaningfully start a new process, so Windows has no visibility on it. However, within Ubuntu, reading the bash history behaves as expected, so there is another record of what commands a user ran.

Amanda Draeger, amanda@tindrasgrove.com
3.1.3. WSL Running PowerShell Commands

WSL allows the execution of any Windows commands, including PowerShell, as long as the user enters the full executable name. For example, the user may run calc.exe to open Windows Calculator, but calc is not a valid command. To execute PowerShell commands from the bash prompt, a user should enter `powershell.exe <command>`. An important item to keep in mind when running commands is the directory the user is in when running them. This is illustrated by the execution of `pwd` to list the present working directory.

![Figure 7. Working directories in Ubuntu and PowerShell from Ubuntu.](image)

The execution of PowerShell from Ubuntu created many entries in the PowerShell/Operational log. Many of these involved the starting of PowerShell itself. Eventually, the actual command created a log entry, shown below.

![Figure 8. PowerShell log entry for pwd command.](image)

This command illustrates an interesting property of PowerShell. The `pwd` command is not a cmdlet; it is an alias for the `Get-Location` cmdlet. The below log entry shows the cmdlet that executed.

Amanda Draeger, amanda@tindrasgrove.com
Looking for Linux: WSL Key Evidence

Figure 9. PowerShell log entry showing Get-Location invocation.

While the PowerShell logs effectively track what eventually executed, they do not do as well with recording the invocation of those commands. Process creation logs are essential to capturing that information. The Sysmon logs do an adequate job of logging both the executed PowerShell command and that the parent process was WSL (log data truncated).

Process Create:
ProcessId: 6560
Image: C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
FileVersion: 10.0.17763.1 (WinBuild.160101.0800)
Description: Windows PowerShell
Product: Microsoft® Windows® Operating System
Company: Microsoft Corporation
OriginalFileName: PowerShell.EXE
CommandLine: C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe pwd
CurrentDirectory: C:\Windows\system32\
User: MSEDGEWIN10\IEUser
ParentProcessId: 7016
ParentImage: C:\Windows\System32\wsl.exe
ParentCommandLine: C:\Windows\system32\wsl.exe ~ -d Ubuntu

The Windows Security process creation logs also record the command and the parent process. However, this log interprets the command from the perspective of WSL. That is, it sees powershell.exe in relation to /mnt/c/ (where the Ubuntu environment can access Windows files) instead of C:\.

Process Information:
New Process ID: 0x1808
New Process Name: C:\Users\IEUser\AppData\Local\Packages\CanonicalGroupLimited.UbuntuonWindows_79rhkp1fndgsc\LocalState\rootfs\init
Creator Process ID: 0x1b10
Creator Process Name: C:\Users\IEUser\AppData\Local\Packages\Canonical

Amanda Draeger, amanda@tindrasgrove.com
PowerShell script block logging behaves similarly to the logging of PowerShell commands run from the Ubuntu environment. That is, the PowerShell/Operational logs record all of the PowerShell activity, but the analyst must look at process creation logs to see that the commands originated from the Ubuntu environment. When executing scripts, it is especially important to remember that the working directory is different. This means altering any relative paths or using an absolute path. Also, Windows and Linux use different slashes in their paths, so quotations should be placed around the path.

PowerShell transcripts work well regardless of what process called PowerShell. The main difference with commands not executed from within PowerShell is that each command is effectively a new PowerShell instance. This means that, when executing commands from the Ubuntu environment, each command will generate its own transcript file.

### 3.1.4. PowerShell Running Linux Commands

As users may run PowerShell commands from the Ubuntu environment, they may also run Linux commands from PowerShell. The user does this by running `wsl <command>`. In contrast to what happens when running commands from the Ubuntu environment, Linux commands run from PowerShell execute from the current Windows directory. This is illustrated below, where the `wsl pwd` command returns the Linux mount point for the same Windows directory as running `pwd` from PowerShell.

Amanda Draeger, amanda@tindrasgrove.com
As these commands originated in PowerShell, the PowerShell/Operational logs capture the exact commands entered, seen in Figure 11 below. The various process creation logs have visibility into the Linux processes, and correctly record PowerShell as the parent process. That is, this behaves consistently with everything learned so far about process creation logging.

### 3.1.5. Daemonized WSL Process

The fact that WSL is not full virtualization or containerization has been previously discussed. As it is essentially emulation, some features of a typical Linux operating system are missing. One of those features is a fully-fledged init system, as demonstrated by the error message below.

```
tindra@MSEDGEWIN10:~$ sudo systemctl start mydaemon.service
System has not been booted with systemd as init system (PID 1). Can't operate.
```

Figure 12. Error message reporting no systemd.
The impact of this is that some persistence mechanisms, such as starting a service or adding a script to execute on boot, will not work. However, it is possible to daemonize a running process. To demonstrate this, the below script was used.

![Figure 13. mydaemon script.]

The `setsid` command was used to daemonize the script as a process.

![Figure 14. Using setsid to daemonize the mydaemon script.]

The `mydaemon` process is visible as a running process within top, as shown below.

![Figure 15. Top showing mydaemon as a running process.]

Sysmon did not record any occurrence of the `mydaemon` process or the commands the script executed. However, the Windows Security process creation logs

Amanda Draeger, amanda@tindrasgrove.com
captured the `mydaemon` process, as well as each execution of `date` and `sleep`, as shown in the next three screenshots.

![Event 4688, Microsoft Windows security auditing.](image1)

*Figure 16. Log recording mydaemon script execution.*

![Event 4688, Microsoft Windows security auditing.](image2)

*Figure 17. Log recording sleep command execution.*

Amanda Draeger, amanda@tindrasgrove.com
Looking for Linux: WSL Key Evidence

The significant difference between this daemonized process and the other processes run from inside the Ubuntu environment is that this process keeps running; it does not execute and exit. This makes it possible to see some evidence of the daemonized process as an actively running process within the Windows Task Manager. Task manager did not have visibility of the mydaemon process, but it recorded the sleep process, as seen below.

Figure 18. Log recording date command execution.

Figure 19. Windows Task Manager showing the WSL sleep process.

Amanda Draeger, amanda@tindrasgrove.com
3.1.6. **Windows Network Connections**

While Windows has many logs that administrators may enable, its network logging leaves something to be desired. Windows Firewall can record network connections, but those records are not in the Event Log format; they are separate files. They also provide only minimal information about network connections. Logging for Domain Name Service (DNS) is significantly better. There is a built-in log for the Windows DNS Client (Draeger, 2019), and Sysmon now includes a DNS log as Event ID 22 (Russinovich & Garnier, 2019). Sysmon also includes a Network Connection (Event ID 3), but these logs do not capture all network-based events. The log below (truncated) shows a ping. Note that the originating process is `svchost.exe`, not `ping.exe`.

Network connection detected:
ProcessId: 1920
Image: C:\Windows\System32\svchost.exe
User: NT AUTHORITY\NETWORK SERVICE

The corresponding Sysmon DNS event correctly identifies the originating process as `ping.exe`. However, this event type will only record DNS events, not all network connections.

Dns query:
ProcessId: 7560
QueryName: tindrasgrove.com
QueryStatus: 0
Image: C:\Windows\System32\PING.EXE

Administrators need to be aware of this lack of visibility on their Windows hosts. Even with extremely verbose host logging, network monitoring is still a valuable tool.

3.1.7. **WSL Network Connections**

In Section 3.1.2, there was a ping from the Ubuntu environment. Below is the corresponding Sysmon Network Connection log (truncated) for the DNS request. The Process ID is the same across the two log entries which confirms that this is the same ping depicted in Figure 6. Neither Sysmon nor Windows DNS Client Logs recorded the DNS request.

Network connection detected:
ProcessId: 6404

Amanda Draeger, amanda@tindrasgrove.com
Looking for Linux: WSL Key Evidence

Image: <unknown process>
User:
Protocol: udp
DestinationPort: 53
DestinationPortName: domain

Note that this Network Connection event only recorded the domain lookup; it did not record the ping itself. To try to generate more network traffic, the system executed `curl www.tindrasgrove.com`. This should generate both the DNS request and the resulting web traffic connection. The below two logs (truncated) show that this was the result.

Network connection detected:
ProcessId: 5560
Image:
C:\Users\IEUser\AppData\Local\Packages\CanonicalGroupLimited.UbuntuonWindows_79rhkp1fndgsc\LocalState\rootfs\usr\bin\curl
User:
Protocol: udp
DestinationPort: 53
DestinationPortName: domain

Network connection detected:
ProcessId: 5560
Image:
C:\Users\IEUser\AppData\Local\Packages\CanonicalGroupLimited.UbuntuonWindows_79rhkp1fndgsc\LocalState\rootfs\usr\bin\curl
User: MSEDGEWIN10\IEUser
Protocol: tcp
DestinationIp: 146.66.99.37
DestinationHostname: ip-146-66-99-37.siteground.com
DestinationPort: 80
DestinationPortName: http

While these logs recorded the process and the destination IP addresses, they do not contain the tindrasgrove.com domain. The Sysmon DNS logs also did not record this event. The Windows DNS Client logs did not record the initial DNS query, but recorded a later stage in the recursive request, as shown below. Based on this, the Windows Process Creation logs are currently the best record of how events executed, even when they are network events.

Amanda Draeger, amanda@tindrasgrove.com
3.1.8. Windows File Access

Auditing Windows file access is a two-step process. The administrator first must enable object access auditing (Microsoft Corporation, 2017). Once this is complete, they must enable auditing for the specific resources they wish to monitor (Microsoft Corporation, 2017). The target.txt file was created to test object auditing. To ensure all possible activity was monitored, all permissions were selected for auditing, and the Principal to monitor was "Authenticated Users." Any accesses of this file were recorded in the Security log with an Event ID 4656, as shown below.

Although all possible permissions were selected for auditing the file, not all activities generated event log entries. For example, Get-ChildItem, which is essentially a directory listing, did not trigger an event, while Get-Content, which reads the contents of the object, did. Additionally, while this event log lists the parent process (as shown below), there is no information that can be used to easily correlate the object access log

Amanda Draeger, amanda@tindrasgrove.com
with PowerShell Operational logs. This makes it difficult to determine with complete certainty that a specified object access event originated from a particular command.

<table>
<thead>
<tr>
<th>Process Information:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process ID:</td>
</tr>
<tr>
<td>Process Name:</td>
</tr>
</tbody>
</table>

### 3.1.9. Access Windows file from WSL

The official Microsoft recommendation for working with files is to store them in the typical Windows file system and access them from either Windows or WSL (Turner R., 2016). Accessing the Windows file system from the Ubuntu environment is straightforward: the Windows drive letters are mounted to /mnt/<drive letter>/.

The below screenshot shows accessing the target.txt file from within the Ubuntu environment using the `cat` command.

![Figure 22. Results of using cat command on a Windows file.](image)

Figure 22. Results of using cat command on a Windows file.

![Event 4656, Microsoft Windows security auditing.](image)

Figure 23. Audit log from using cat command on a Windows file.

Amanda Draeger, amanda@tindrascroge.com
This log entry correctly interprets the process that accessed the file as cat. However, this does not hold true for all methods of accessing a file. When the touch command was used, the audit log entry recorded the SearchProtocolHost.exe process, not touch, as seen below. This is because the Linux touch command does not know how to modify attributes on a Windows file, so a Windows command must be called to perform this function instead.

![Figure 24. Audit log from using touch command on a Windows file.](image)

### 3.1.10. Access WSL file from Windows

On versions of Windows before version 1903, directly accessing WSL’s file system from Windows was not advised (Turner R., 2016). However, as it is possible, it is still worth understanding. The location of the Ubuntu environment’s files can be found by examining any of the log files that identify a Linux executable, such as the log from Figure 23. On the test VM, that location is `C:\Users\IEUser\AppData\Local\Packages\CanonicalGroupLimited.UbuntuonWindows_79rhkp1fndgsc\LocalState\rootfs`. The rootfs folder is equivalent to `/` within the Ubuntu environment. The below screenshot shows the contents of the Ubuntu user tindra’s home folder, as seen from Windows Explorer.
Looking for Linux: WSL Key Evidence

The contents of testfile are shown below. This file had the same auditing configuration applies as target.txt.

![Linux user's home folder as viewed from windows explorer.](image)

**Figure 25.** Linux user’s home folder as viewed from windows explorer.

The contents of testfile are shown below. This file had the same auditing configuration applies as target.txt.

```bash
$ cat testfile
Tindra@MSEWIN10:$ cat testfile.
This file is on Ubuntu.
Tindra@MSEWIN10:$
```

**Figure 26.** Contents of testfile.

The below log entry (truncated) was created by the above reading of the file from the Ubuntu environment. It correctly interprets the file location and process, as Windows sees them.

```
Object:
  Object Type:                File
  Object Name:                C:\Users\IEUser\AppData\Local\Packages\CanonicalGroupLimited.UbuntuonWindows_79rhkp1fndgsc\LocalState\rootfs\home\tindra\testfile
  Process Information:
    Process ID:                0x12ac
    Process Name:                C:\Users\IEUser\AppData\Local\Packages\CanonicalGroupLimited.UbuntuonWindows_79rhkp1fndgsc\LocalState\rootfs\bin\cat
```

The below log (truncated) shows the same file being opened using Notepad.

```
Object:
  Object Server:                Security
  Object Name:                C:\Users\IEUser\AppData\Local\Packages\CanonicalGroupLimited.UbuntuonWindows_79rhkp1fndgsc\LocalState\rootfs\home\tindra\testfile
```

Amanda Draeger, amanda@tindrasgrove.com
Looking for Linux: WSL Key Evidence

Process Information:

Process ID: 0xfd8
Process Name: C:\Windows\System32\notepad.exe

Although not recommended, editing testfile with Notepad and saving those changes was successful. The contents of the modified testfile, as read from within the Ubuntu environment, are shown below.

![Figure 27. Contents of modified testfile.](image)

This means that administrators can read any files within the Ubuntu environment. This has positive implications for the ability to directly read log files, or the bash history file, as seen below.

![Figure 28. Windows Notepad reading .bash_history file.](image)

3.2. Synthesis

To demonstrate a hypothetical real-world event, some sample code from Cleary (2018) was run. The code retrieves a file from a web source and loads it into the current PowerShell session. The SecureString (AES) Encoding example was saved to the Windows file system and named Obfuscate2.ps1. This script was then run from within the Ubuntu environment, as shown below.

![Figure 29. Execution of Obfuscation2.ps1 from Ubuntu environment.](image)

Amanda Draeger, amanda@tindrasgrove.com
Both Sysmon and Windows Security logs had visibility of the process creation, as shown below.

![Figure 30. Windows Process creation log for Obfuscate2.ps1.](image)

The Windows DNS Client and Sysmon DNS logs both observed the DNS connections to the domain in the script, as seen below. Note that the DNS logs report the decimal PID, while the Security log reports the PID in hexadecimal.

![Figure 31. DNS Client log for Obfuscate2.ps1.](image)
PowerShell/Operational logs provided a significant amount of detail regarding the activity. Most notable is the fact that while the Obfuscate2.ps1 script contained obfuscated code, the script block logging translated that into the code that ran on the system.

![Figure 32. PowerShell script block log for Obfuscate2.ps1](image)

By enabling the various available log types, security analysts can walk a single event through multiple data sources to gain a full picture of what activity actually took place. A summary of the log types discussed in this paper is shown in Figure 33.

<table>
<thead>
<tr>
<th>Log Source</th>
<th>Event ID</th>
<th>Log Contents</th>
<th>Usefulness</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microsoft Windows DNS Client Events/Operational</td>
<td>3006</td>
<td>DNS Client Request</td>
<td>Sometimes</td>
</tr>
<tr>
<td>Microsoft Windows Powershell/Operational</td>
<td>4103</td>
<td>PowerShell command invocation</td>
<td>Sometimes</td>
</tr>
<tr>
<td>Microsoft Windows Powershell/Operational</td>
<td>4104</td>
<td>PowerShell Module and Script Block logging</td>
<td>Very</td>
</tr>
<tr>
<td>PowerShell Transcripts in C:\Users&lt;UserName&gt;\Documents&lt;Date&gt;</td>
<td>N/A</td>
<td>Readout of what got printed to screen</td>
<td>Very</td>
</tr>
<tr>
<td>Security</td>
<td>4688</td>
<td>Process Creation</td>
<td>Very</td>
</tr>
<tr>
<td>Security</td>
<td>4656</td>
<td>Object Access</td>
<td>Very</td>
</tr>
<tr>
<td>Sysmon/Operational</td>
<td>1</td>
<td>Process Creation</td>
<td>Sometimes</td>
</tr>
<tr>
<td>Sysmon/Operational</td>
<td>5</td>
<td>Process Termination</td>
<td>Sometimes</td>
</tr>
<tr>
<td>Sysmon/Operational</td>
<td>22</td>
<td>DNS Requests</td>
<td>Sometimes</td>
</tr>
<tr>
<td>Sysmon/Operational</td>
<td>3</td>
<td>Network Connection</td>
<td>Rarely</td>
</tr>
</tbody>
</table>

![Figure 33. Summary of log sources and their contents.](image)

### 3.3. Version Notes

The “MS Edge Test” VM was Windows version 1809. Version 1903 made significant changes to WSL. Most notable among these changes is the ability to interact with WSL’s file system using `\wsl$<DistroName>` (Turner R., 2016). Administrators

Amanda Draeger, amanda@tindrasgrove.com
must understand that updates can significantly change system behavior, so they must test any signatures they have developed for system monitoring with each updated version.

All of this research applies only to WSL Version 1. The upcoming release of WSL Version 2 is projected to use very different methods to achieve the running of Linux executables from Windows. This means that many of the behaviors identified in this research will not apply to that version.

4. Recommendations and Implications

4.1. Recommendations for Practice

When deciding on their monitoring strategy, organizations should first look at their policy. Is WSL authorized in their environment? Is such use restricted to developers and other non-typical users? The answers to these questions will guide what activities are most important to them so that they can tune and filter log collection appropriately.

If WSL use is authorized, organizations need to ensure that they have compensated for their environment. The exact distribution and version of Linux installed may significantly change some behaviors. Upgrading to the latest patch level of Windows will change some behaviors. Commercially available endpoint solutions may have different visibility into how WSL interacts with the underlying Windows operating system. As with any other piece of software, having a standardized baseline across the organization can make life easier for both administrators and security analysts.

Administrators need to understand that WSL is not a full Linux system. Actions that they take to influence and monitor behavior on other Linux systems may not work on WSL. This is especially true in the case of any services or startup scripts. Blind spots may result from a failure to adapt controls to the different behavior of WSL.

4.2. Implications for Future Research

As the Microsoft team continues to update and improve WSL, research will need to continue to determine how to document and monitor its behavior. Also, some areas of Windows logging could use focus. In particular, logging of network connections, even with Sysmon, is far from complete at this time. Better documentation of how programs

Amanda Draeger, amanda@tindrasgrove.com
interact with the Windows network stack would assist security analysts in identifying anomalous behavior.

5. Conclusion

Before now, there has been little research into the behavior of WSL. This research project is far from comprehensive, but it focuses on key behaviors that correlate to techniques in MITRE’s ATT&CK. Process creation, network connections, and file access are critical activities for defenders to be able to monitor. While this research captures the current state of these behaviors with WSL, as updates are made to WSL and Windows, it will need to be updated. Regardless of the challenges of essentially running two operating systems on the same computer, organizations can gain fairly complete visibility into everything their computers are doing with the use of native Windows logging.
References


Microsoft Corporation. (2017, July 24). Apply a basic audit policy on a file or folder. Retrieved from Microsoft Docs: https://docs.microsoft.com/en-

Amanda Draeger, amanda@tindrasgrove.com
Looking for Linux: WSL Key Evidence

us/windows/security/threat-protection/auditing/apply-a-basic-audit-policy-on-a-file-or-folder


Amanda Draeger, amanda@tindrasgrove.com
## Upcoming SANS Training

Click here to view a list of all SANS Courses

<table>
<thead>
<tr>
<th>Event Name</th>
<th>Location</th>
<th>Dates</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>SANS Northern VA - Reston Spring 2020</td>
<td>Reston, VAUS</td>
<td>Mar 02, 2020 - Mar 07, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Munich March 2020</td>
<td>Munich, DE</td>
<td>Mar 02, 2020 - Mar 07, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Secure Japan 2020</td>
<td>Tokyo, JP</td>
<td>Mar 02, 2020 - Mar 14, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS St. Louis 2020</td>
<td>St. Louis, MOUS</td>
<td>Mar 08, 2020 - Mar 13, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Dallas 2020</td>
<td>Dallas, TXUS</td>
<td>Mar 09, 2020 - Mar 14, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Prague March 2020</td>
<td>Prague, CZ</td>
<td>Mar 09, 2020 - Mar 14, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>Wild West Hackin Fest 2020</td>
<td>San Diego, CAUS</td>
<td>Mar 10, 2020 - Mar 11, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Doha March 2020</td>
<td>Doha, QA</td>
<td>Mar 14, 2020 - Mar 19, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Norfolk 2020</td>
<td>Norfolk, VAUS</td>
<td>Mar 16, 2020 - Mar 21, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Kuwait March 2020</td>
<td>Salmiya, KW</td>
<td>Mar 21, 2020 - Mar 26, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS 2020</td>
<td>Orlando, FLUS</td>
<td>Apr 03, 2020 - Apr 10, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Muscat April 2020</td>
<td>Muscat, OM</td>
<td>Apr 04, 2020 - Apr 09, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Riyadh April 2020</td>
<td>Riyadh, SA</td>
<td>Apr 04, 2020 - Apr 16, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Bethesda 2020</td>
<td>Bethesda, MDUS</td>
<td>Apr 14, 2020 - Apr 19, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Minneapolis 2020</td>
<td>Minneapolis, MNUS</td>
<td>Apr 14, 2020 - Apr 19, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Brussels April 2020</td>
<td>Brussels, BE</td>
<td>Apr 20, 2020 - Apr 25, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>Blue Team Summit &amp; Training 2020</td>
<td>OnlineKYUS</td>
<td>Mar 02, 2020 - Mar 09, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS OnDemand</td>
<td>Books &amp; MP3s OnlyUS</td>
<td>Anytime</td>
<td>Self Paced</td>
</tr>
</tbody>
</table>