Global Information Assurance Certification Paper

Preventing Living off the Land Attacks

GIAC (GPEN) Gold Certification

Author: David Brown, mrdavebrown@gmail.com
Advisor: *David Fletcher*

Accepted: *January 6, 2020*

Abstract

Increasingly, attackers are relying on trusted Microsoft programs to carry out attacks against individuals and organizations (Symantec, 2017). The software typically comes installed by default in Windows and is often required for the essential functionality of the operating system. These types of attacks are called "living off the land," and they can be challenging to detect and prevent. This paper examines the viability of using Microsoft AppLocker to thwart living off the land attacks without impacting the legitimate operating system and administrative use of the underlying Microsoft programs.

1. Introduction

Built-in Microsoft Windows programs are increasingly becoming cybercriminals' go-to tools for perpetuating attacks (Symantec, 2017). These attacks, called living off the land attacks, are ideal for cybercriminals for several reasons. One reason is that attackers may not need to place foreign, malicious tools on the target machines. Many of the tools needed to perpetrate attacks are built-in to Windows by default. Since they are trusted, built-in Microsoft programs, they can also be used to bypass anti-virus and traditional application whitelisting deployments (Carbon Black, 2019). Another reason is that attackers' use of the built-in Microsoft programs can be difficult to distinguish from the legitimate operating system and administrative use of the built-in programs. This allows attackers to comingle their activities amongst regular system and administrative logs and thus hide their activity (Rapid7, 2019).

The quantity of living off the land attacks are increasing. CrowdStrike stated that "LOTL (living off the land) tactics, which do not involve malware, have picked up significantly in the world of cyber espionage in recent years" (CrowdStrike, 2019). A CrowdStrike report went on to assert that 40% of all global attacks they observed in 2018 were malware-free, meaning that they relied entirely on built-in programs (CrowdStrike, 2019). Living off the land attacks also pose a unique security challenge in that they are often able to bypass traditional application whitelisting deployments. Application whitelisting is held up as a gold standard in preventing malicious attacks (Australian Cyber Security Centre, 2019). However, a quick peruse through the MITRE ATT&CK Framework and the LOBAS Project reveals many built-in Microsoft programs that can be used specifically to evade application whitelisting (MITRE, n.d.) (LOBAS, n.d.).

The challenge of these powerful built-in Microsoft programs is that the Windows operating system and administrators legitimately use them and that cybercriminals maliciously abuse them. Many modern tools, such as Windows Defender, seek to solve the problem by analyzing the behavior of the built-in programs. If the built-in program behaves in a particular way that is known to be malicious or seems to be malicious, Windows Defender blocks the action. While this is a significant step forward, the malicious behavior would still be allowed to execute if it behaved in a way that did not

appear to Windows Defender as malicious. Another solution must be found to truly prevent living off the land attacks.

This research seeks to develop ways to prevent living off the land attacks using application whitelisting. The application whitelisting techniques used in this paper specifically utilize Microsoft's AppLocker in a user-aware context. By creating user-based rules to block built-in Microsoft programs, legitimate operating system and administrative use of the programs should not be impacted, but regular users' use of the abused Microsoft programs should be prevented. Thus, when cybercriminals take over a regular user account, they will be prevented from executing the programs.

2. Research Method

A fully patched Windows 10 Enterprise virtual machine is configured with three regular users. The first user, named Regular, is set up as a regular user with no AppLocker restrictions. The second user, named Default, is set up as a regular user with default AppLocker rules enforced. The default AppLocker rules are created by selecting the "Create Default Rules" option for Executable Rules, Windows Installer Rules, Script Rules, and Packaged app Rules. These rules can be referenced in Appendix C (AppLocker Default Rules). The third user, named LotL, is set up as a regular user with default AppLocker rules enforced plus additional AppLocker rules designed to prevent living off the land attacks. These rules are referred to in the paper as LotL rules and can be found in Appendix B (AppLocker LotL Rules). The user LotL is also given group membership to a security group called Employees. At the foundational level, the LotL rules are designed to restrict members of the Employees group from executing built-in Windows programs that are abused by attackers. The LotL rules only add restrictions for members of the Employees group. This distinction allows administrators and the operating system to maintain access to the built-in Windows programs. For additional details about the virtual machine, see Appendix A (Virtual Machine Setup).

The Findings and Discussion section of this paper explores various attacks that utilize built-in Windows programs to perpetrate attacks. The attacks are run as the user Regular to observe the success of the attacks without AppLocker. The attacks are then

run as the user Default to observe the success of the attacks with a default AppLocker deployment. Finally, the attacks are run as the user LotL to observe if the LotL rules can successfully thwart the demonstrated living off the land attacks.

AppLocker was used instead of Windows Defender Application Control since AppLocker is user-aware. The user-aware feature is essential in allowing built-in Microsoft programs to be used by the operating system and administrators while blocking the programs for regular users. If the blocks were applied within Windows Defender Application Control without user-awareness, legitimate operating system and administrative functionality would be impaired. According to Microsoft, AppLocker should be used when "you need to apply different policies for different users or groups on a shared computer" (Microsoft, 2019, January 1). Microsoft goes on to recommend that, an ideal, real-world deployment solution would use Windows Defender Application Control for the traditional (base) whitelisting functionality and AppLocker for blocking programs that need to be user-aware (Microsoft, 2019, January 1).

3. Findings and Discussion

This section explores three types of attacks that are being used in the wild to bypass traditional application whitelisting deployments. These attacks demonstrate AppLocker's ability to block living off the land attacks. There are many more attack types than just these three. Other illustrative attacks could easily be swapped for the ones presented in this section. The LOBAS Project, for example, lists over 100 Microsoft programs that can be used in living off the land attacks (LOBAS, n.d.).

3.1. Code Execution with JavaScript and Visual Basic

3.1.1. Demonstrating Attacks

An attacker can arbitrarily execute unsigned code on a Windows machine by dropping a JavaScript file (JS) or a Visual Basic file (VBS) and then executing the file with cscript.exe or wscript.exe.

The JavaScript code to demonstrate this is as follows:

```
var cmd = new ActiveXObject("WScript.Shell");
cmd.run("powershell Test-Connection 127.0.0.1 -Count 10 | ft address,
        responsetime");
```

The Visual Basic code to demonstrate this is as follows:

```
set cmd = CreateObject("WScript.Shell")
cmd.run("powershell Test-Connection 127.0.0.1 -Count 10 | ft address,
        responsetime")
```

On the virtual machine, the above code can be placed in files called "wscript_powershell_ping.js" and "wscript_powershell_ping.vbs" respectively. The code can be executed by double-clicking the files or via cscript.exe or wscript.exe. See Figure 1 for example output.

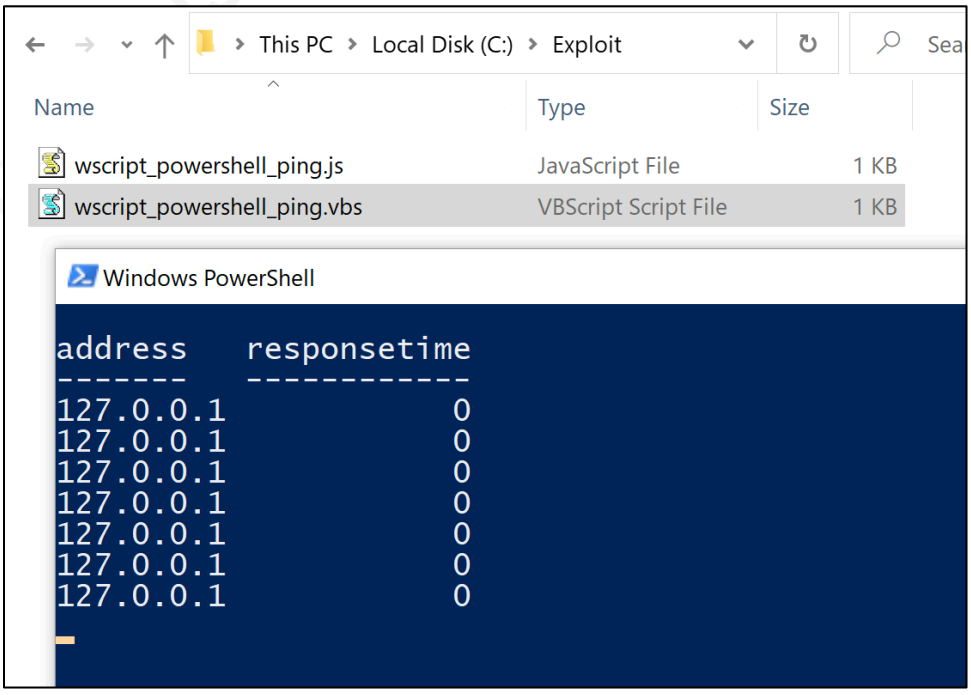


Figure 1. Successful VBS script execution via wscript.exe.

If AppLocker is deployed on the workstation with default settings, only scripts that reside in the Program Files folder or the Windows folder would be allowed to

execute. In this situation, the code execution would fail, and the following error in Figure 2 would appear:

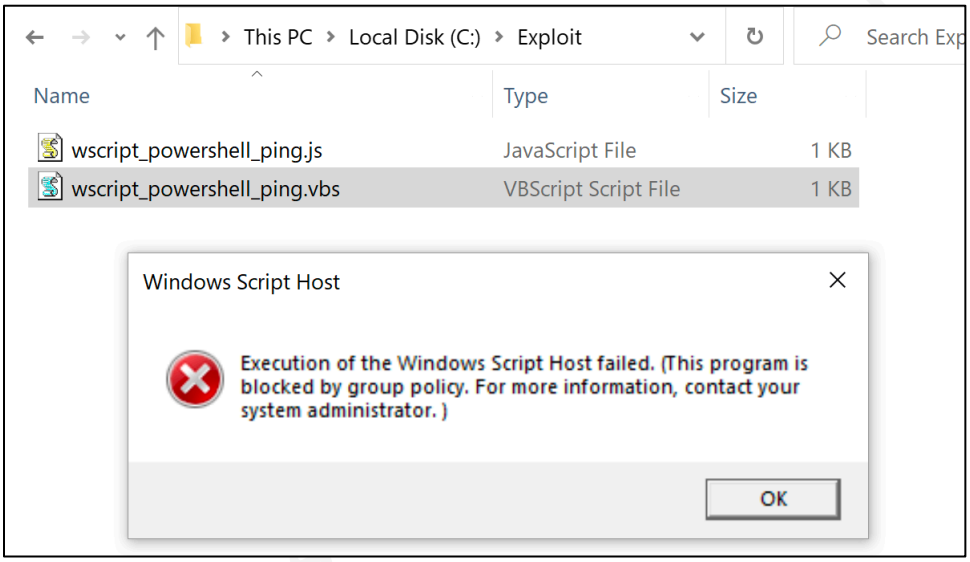


Figure 2. Block of VBS script due to default AppLocker rules.

Figure 3 shows the Event Viewer default block under Applications and Services, Microsoft, Windows, AppLocker, MSI and Script, Event ID 8007.

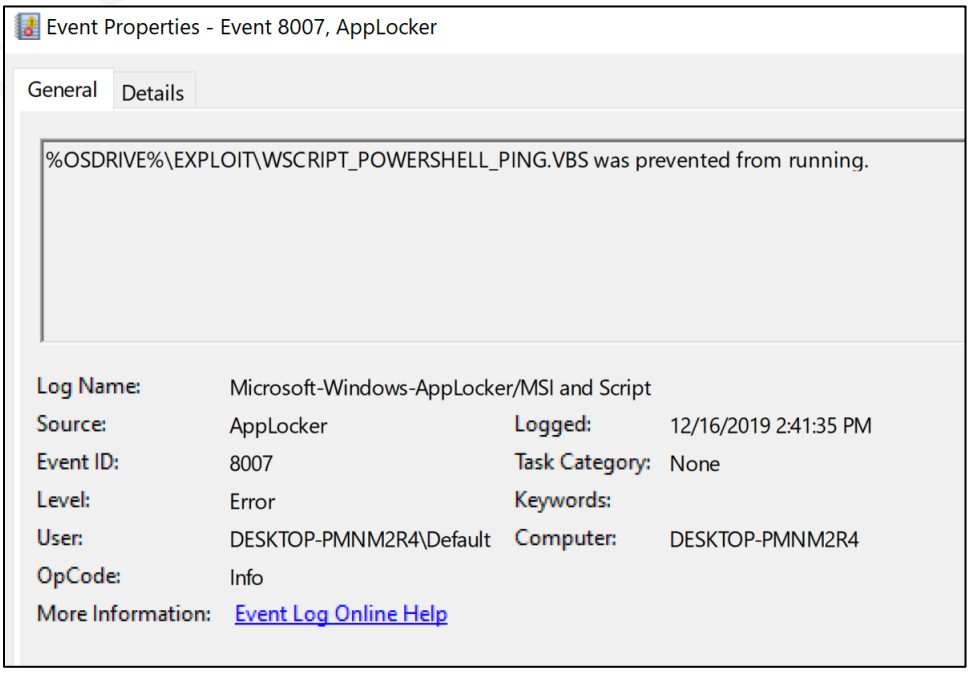


Figure 3. Event Viewer details of AppLocker VBS script block.

David Brown, mrdavebrown@gmail.com

One way to bypass AppLocker's restrictions on scripts is to use Microsoft HTML Applications (HTA). HTAs are standalone applications that execute outside of a browser, thus bypassing browser security settings. Mshta.exe is a built-in Microsoft program that executes HTA files (MITRE, n.d.). HTA files can be downloaded from the Internet, received in a malicious email attachment, or crafted on a user's computer. The following HTA file is crafted to launch PowerShell and ping the localhost (Graham, 2018):

```
<script LANGUAGE="VBScript">
Set cmd = CreateObject("WScript.Shell")
cmd.run("powershell Test-Connection 127.0.0.1 -Count 10 | ft address,
        responsetime")
</script>
```

On the virtual machine, the above code is placed in an HTA file called "mshta_powershell_ping.hta". When executed on the virtual machine with default AppLocker rules enabled, the HTA file successfully launches a PowerShell window and begins pinging the localhost. See Figure 4 for example output.

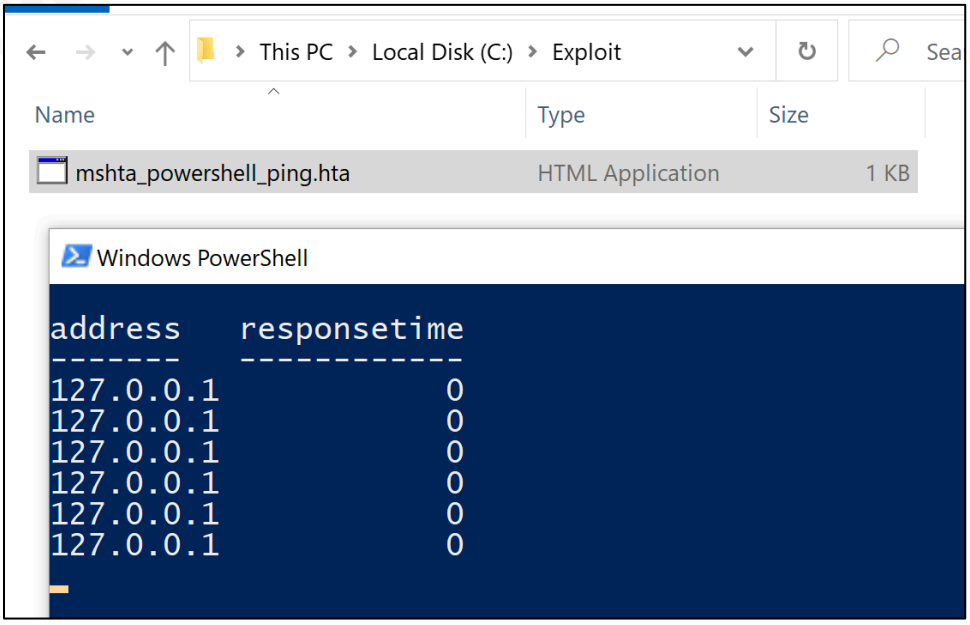


Figure 4. Successful bypass of default AppLocker script rules with HTA execution via mshta.exe.

David Brown, mrdavebrown@gmail.com

3.1.2. Blocking Attacks with LotL Rules

When the LotL rules are enabled, and mshta_powershell_ping.hta executes, the attack fails due to a specific LotL Rule that restricts the execution of mshta.exe. This can be seen in Figure 5.

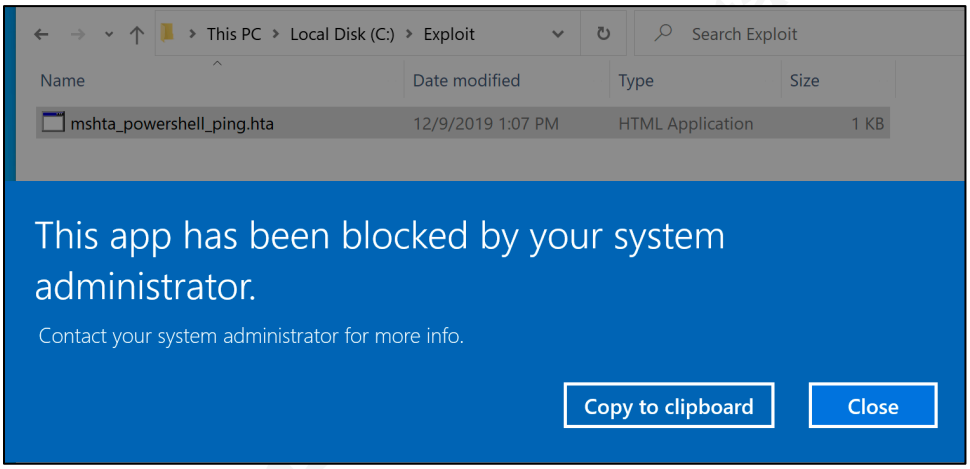


Figure 5. Block of HTA execution due to AppLocker LotL rules.

Figure 6 shows the Event Viewer LotL block under Applications and Services, Microsoft, Windows, AppLocker, EXE and DLL, Event ID 8004.

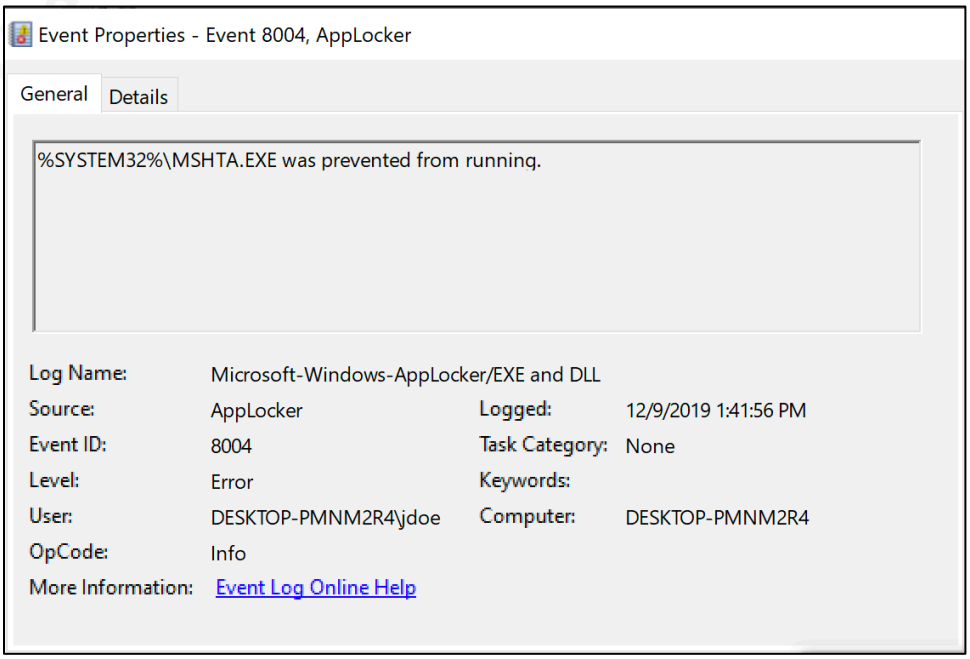


Figure 6. Event Viewer details of AppLocker mshta.exe block.

The first two JavaScript and Visual Basic attacks also fail with the LotL rules enabled. When double-clicking on “wscript_powershell_ping.js” or “wscript_powershell_ping.vbs”, the LotL rules block wscript.exe. In fact, the LotL rules block the attack before the default AppLocker script rules have a chance to evaluate, as seen in Figure 7 and Figure 8.

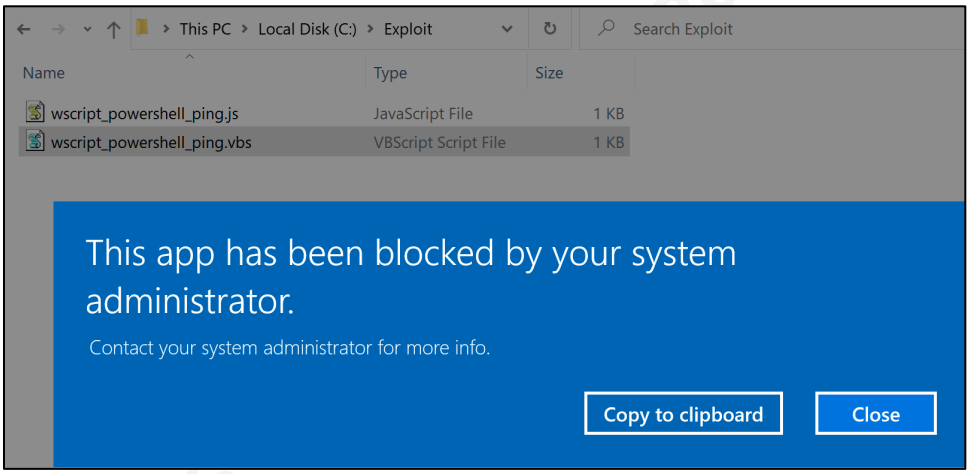


Figure 7. Block of VBS script due to AppLocker LotL rules.

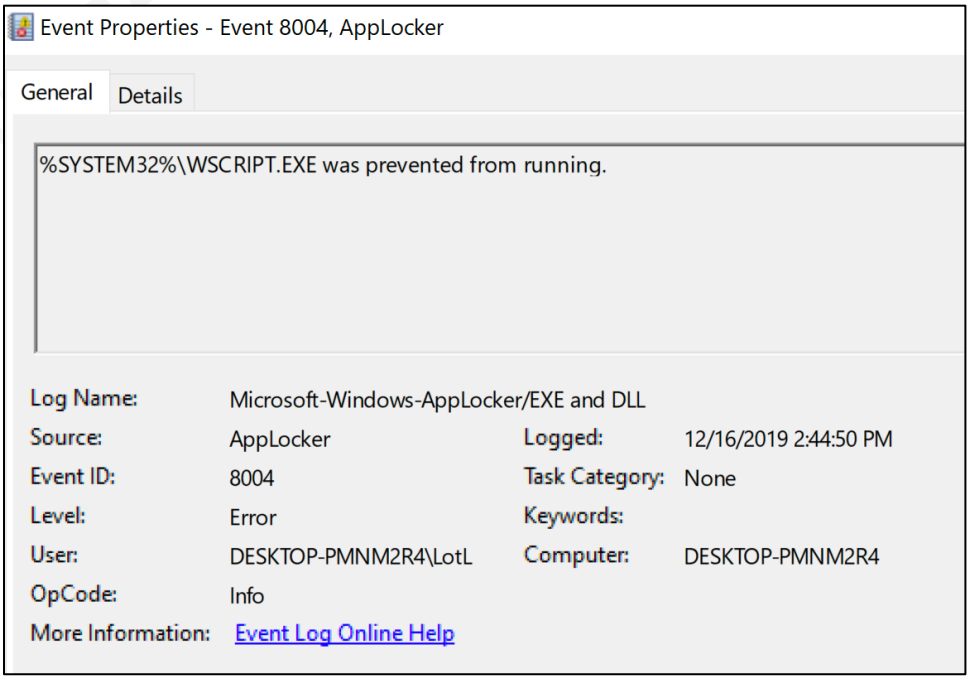


Figure 8. Event Viewer details of wscript.exe block.

3.1.3. Summary of Behavior

The following chart (Table 1) summarizes how the attack demonstrations behaved for each user. For reference, the Regular user does not have AppLocker enabled. The Default user has the default AppLocker rules enabled and a listing of these rules is located in Appendix C (AppLocker Default Rules). The LotL user has the custom AppLocker LotL rules enabled and a full listing of these rules is provided in Appendix B (AppLocker LotL Rules).

Demonstration of Attack	Regular User	Default User	LotL User
wscript_powershell_ping.js	Executed	Blocked JS file	Blocked wscript.exe
wscript_powershell_ping.vbs	Executed	Blocked VBS file	Blocked wscript.exe
mshta_powershell_ping.hta	Executed	Executed	Blocked mshta.exe

Table 1

See Section 3.1.5 for the implementation of the specific LotL rules that relate to the above attacks.

3.1.4. Real-World Examples

A variant of ransomware called RAA demonstrates a real-world example of this attack. RAA is written purely in JScript and delivered to victims via email attachments. When a user opens the attachment, wscript.exe automatically launches and executes the JScript within the JS file. From there, the RAA ransomware is released on the system (Trend Micro, 2016). With AppLocker LotL rules enabled, the RAA attack fails when AppLocker prevents wscript.exe from running.

A variant of ransomware called Sodinokibi is another real-world example. Described by security researchers as “highly evasive,” Sodinokibi uses various built-in Windows programs to perpetuate a ransomware attack. The attack begins a maliciously crafted JavaScript file delivered over email. As seen in the RAA attack, when a user double-clicks the attachment, wscript.exe automatically launches and executes the JavaScript within the JS file. From there, powershell.exe is automatically called to

execute a malicious PowerShell script, and attempts privilege escalation through modifying the registry and launching compmgmtlauncher.exe (Cybereason Nocturnus, 2019). With AppLocker LotL rules enabled, this attack fails when AppLocker blocks the initial wscript.exe execution as well as the subsequent powershell.exe execution.

3.1.5. Specific LotL rules to Prevent the Attacks

To implement the specific AppLocker LotL rules that prevent the above attacks, create the following AppLocker Executable rules set to Deny for "Employees":

Publisher: O=MICROSOFT CORPORATION, L=REDMOND,
S=WASHINGTON, C=US
Product name: MICROSOFT® WINDOWS® OPERATING SYSTEM
File name: POWERSHELL.EXE
File version: *

Publisher: O=MICROSOFT CORPORATION, L=REDMOND,
S=WASHINGTON, C=US
Product name: MICROSOFT® WINDOWS® OPERATING SYSTEM
File name: POWERSHELL_ISE.EXE
File version: *

Publisher: O=MICROSOFT CORPORATION, L=REDMOND,
S=WASHINGTON, C=US
Product name: MICROSOFT ® WINDOWS SCRIPT HOST
File name: CSCRIPT.EXE
File version: *

Publisher: O=MICROSOFT CORPORATION, L=REDMOND,
S=WASHINGTON, C=US
Product name: MICROSOFT ® WINDOWS SCRIPT HOST
File name: WSCRIPT.EXE
File version: *

Publisher: O=MICROSOFT CORPORATION, L=REDMOND,
S=WASHINGTON, C=US
Product name: INTERNET EXPLORER
File name: MSHTA.EXE
File version: *

3.2. Code Execution with .NET Applications

3.2.1. Demonstrating Attacks

.NET applications are an essential component of Microsoft Windows. Installutil.exe, for example, is a built-in Microsoft program that allows for the installation of resources via the .NET Framework. Installutil.exe allows attackers to execute untrusted code via the trusted Microsoft program (MITRE, n.d.). The attack discussed in this section is based on an attack demonstrated by Black Hills Information Security that leverages installutil.exe (Fehrman, 2016). If application whitelisting is not blocking malicious uses of installutil.exe, the attacker can run a malicious PowerShell script.

To carry out the attack, download the powerup.ps1 script from PowerShellMafia to the C:\Exploit folder. Powerup.ps1 is a PowerShell script that looks for Windows privilege escalation vectors (PowerShellMafia, n.d.). Add a line to the end of powerup.ps1 to automatically call the function "Invoke-AllChecks."

```
Invoke-AllChecks -Verbose | Out-File C:\Exploit\allchecks.txt
```

To begin the attack, compile program.cs found in Appendix D. One way to compile the code is to use Microsoft's built-in .NET compiler, csc.exe.

```
C:\Windows\Microsoft.NET\Framework\v4.0.30319\csc.exe
        /r:C:\Windows\assembly\GAC_MSIL\System.Management.Automation\1.
        0.0.0__31bf3856ad364e35\System.Management.Automation.dll /unsafe
        /platform:anycpu /out:C:\Exploit\powerup.exe C:\Exploit\Program.cs
```

Create a shortcut to installutil.exe in the C:\Exploit folder and add the following code after installutil.exe in the "Target" of the shortcut:

```
/logfile=C:\Exploit\log.txt /LogToConsole=false /U
```

Now drag powerup.exe into the installutil.exe shortcut. The attack is successful even if the default AppLocker rules are enabled. The default AppLocker executable rules are bypassed. Microsoft Defender also fails to stop this attack as demonstrated in Figure 9.

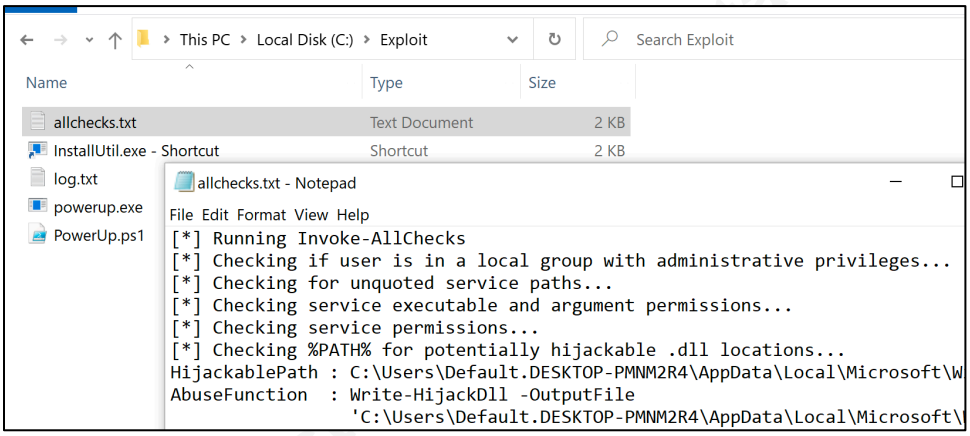


Figure 9. Successful bypass of default AppLocker executable rules with powerup.exe via installutil.exe.

3.2.2. Blocking Attacks with LotL Rules

When LotL rules are enabled, the attack fails. The LotL rules blocked the attack by restricting regular users from running installutil.exe as seen in Figure 10 and Figure 11.

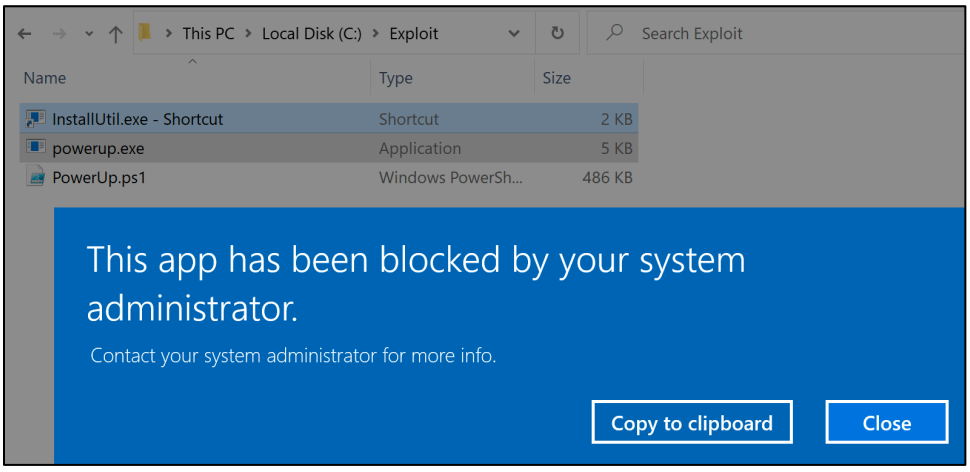


Figure 10. Block of installutil.exe due to AppLocker LotL rules.

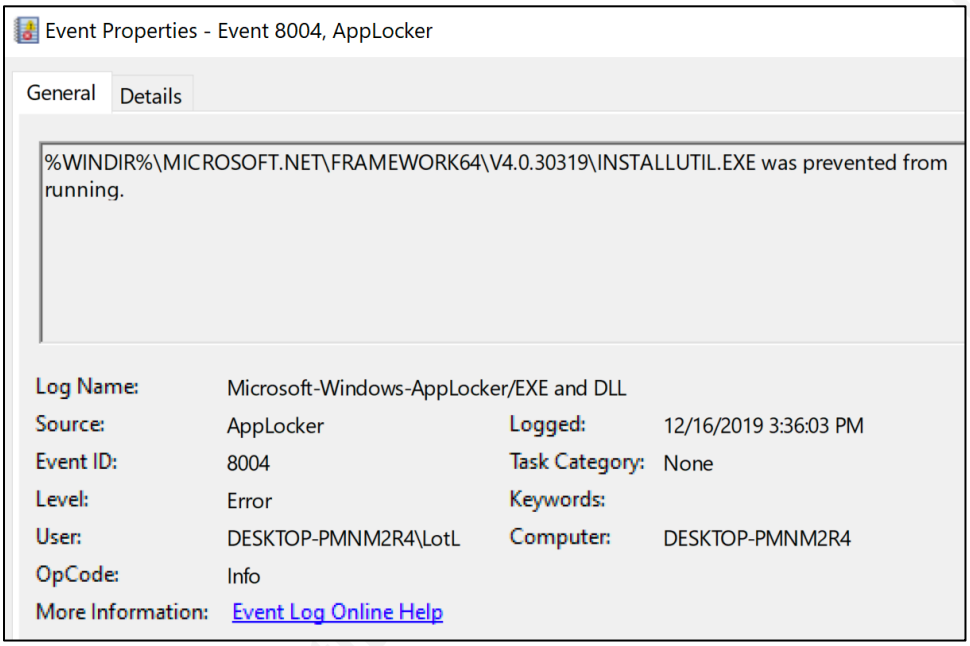


Figure 11. Event Viewer details of installutil.exe block.

Understanding the steps in the kill chain for each living off the land attack is essential in building a layered approach to LotL whitelisting rules. If the LotL rules were not restricting installutil.exe, for example, the attack would fail at csc.exe. While ideal, building whitelisting rules around each built-in Windows program that attackers are leveraging may not always be achievable. An understanding of the steps in each kill chain can help ensure that a LotL rule disrupts at least part of the attack.

If the LotL rule were not blocking installutil.exe, the attack would still fail at the execution of csc.exe. This can be seen in Figure 12 and Figure 13.

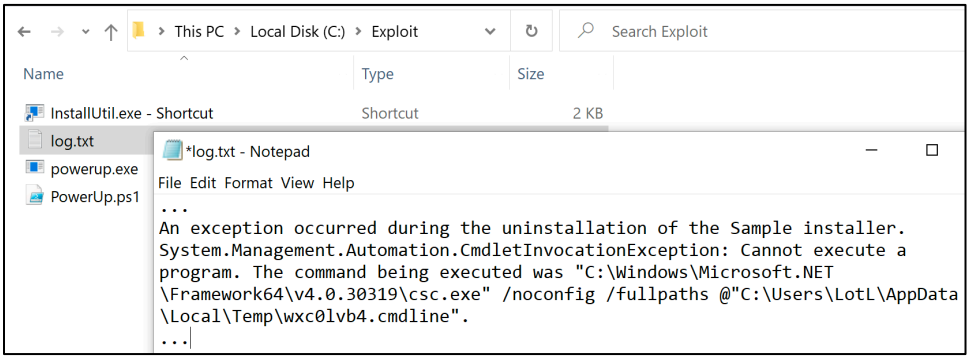


Figure 12. Block of csc.exe due to AppLocker LotL rules.

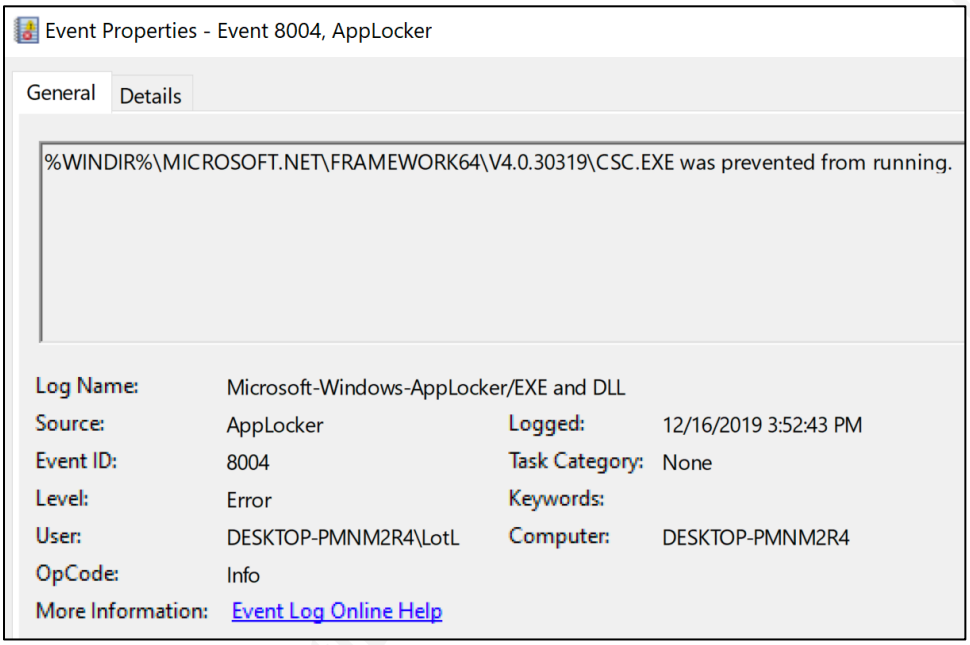


Figure 13. Event Viewer details of csc.exe block.

If both the LotL rules did not restrict installutil.exe and csc.exe, the attack would partially fail at sc.exe as demonstrated in Figure 14 and Figure 15.

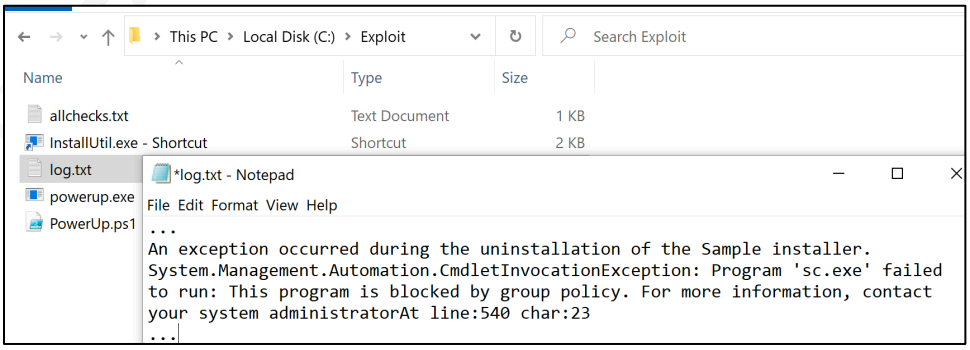


Figure 14. Block of sc.exe due to AppLocker LotL rules.

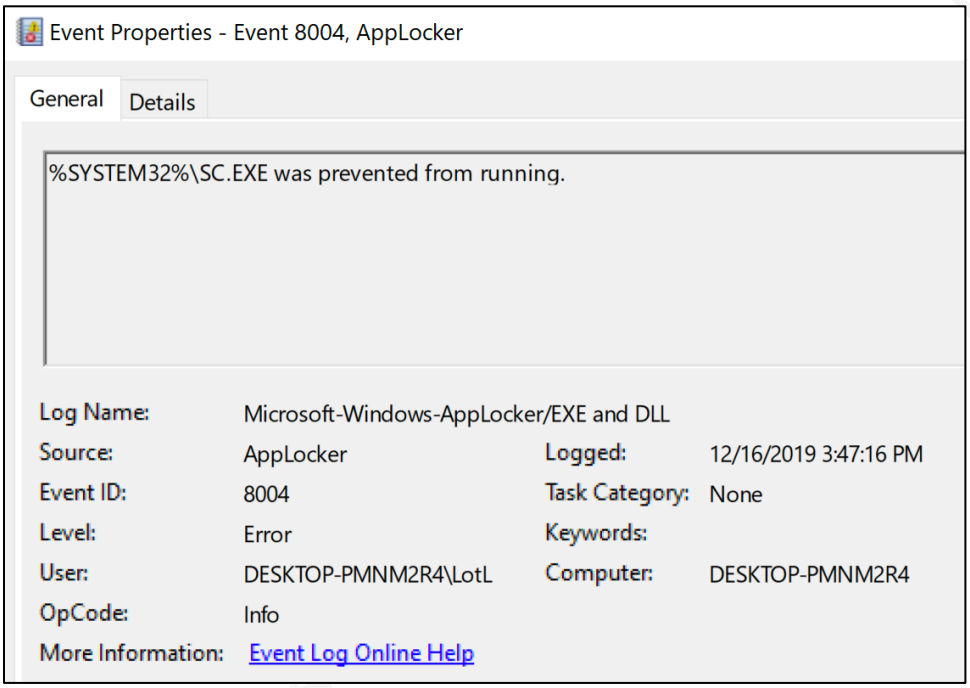


Figure 15. Event Viewer details of sc.exe block.

Understanding the built-in Microsoft tools that are maliciously used becomes vital to disrupting the attack. Each layer of the attack that is stopped prevents lower layers from executing. Monitoring logs also presents an opportunity for defenders to be alerted of the attack as it is in progress. If sc.exe, for example, were the only Microsoft program that was being restricted by whitelisting, that would be enough to alert a defender of the attack. However, it would not be enough to block the attack fully.

3.2.3. Summary of Behavior

The following chart (Table 2) summarizes how the attack demonstrations behaved for each user. For reference, the Regular user does not have AppLocker enabled. The Default user has the default AppLocker rules enabled and a listing of these rules is located in Appendix C (AppLocker Default Rules). The LotL user has the custom AppLocker LotL rules enabled and a full listing of these rules is provided in Appendix B (AppLocker LotL Rules).

Demonstration of Attack	Regular User	Default User	LotL User
powerup.ps1 via powershell.exe	Executed	Blocked PS file	Blocked powershell.exe
powerup.exe via installutil.exe	Executed	Executed	Blocked installutil.exe

Table 2

See Section 3.2.5 for the implementation of the specific LotL rules that relate to the above attacks.

3.2.4. Real-World Examples

APT10 carried out real-world attacks using similar tactics. APT10 used .NET applications to install the QuasarRAT. Specifically, Quasar was installed on the victim machines using installutil.exe and a custom DLL compiled with .NET (PwC, 2017). The AppLocker LotL rules stop the attack at the initial installutil.exe execution.

Sequre Ransomware is another example of a real-world attack that uses .NET applications to perpetuate an attack. In the case of Sequre, csc.exe compiled and executed malicious C# code directly on the victims' machines. The Sequre Ransomware attack compiles and runs in memory, making the malicious activity much harder to detect (Vipre, 2018). With AppLocker LotL rules enabled, csc.exe would be blocked, and the attack would fail.

3.2.5. Specific LotL rules to Prevent the Attacks

To implement the specific AppLocker LotL rules that prevent the above attacks, create the following AppLocker Executable rules set to Deny for "Employees":

Publisher: O=MICROSOFT CORPORATION, L=REDMOND,
S=WASHINGTON, C=US
Product name: MICROSOFT® WINDOWS® OPERATING SYSTEM
File name: POWERSHELL.EXE
File version: *

Publisher: O=MICROSOFT CORPORATION, L=REDMOND,
S=WASHINGTON, C=US
Product name: MICROSOFT® WINDOWS® OPERATING SYSTEM
File name: POWERSHELL_ISE.EXE
File version: *

Publisher: O=MICROSOFT CORPORATION, L=REDMOND,
S=WASHINGTON, C=US
Product name: MICROSOFT® .NET FRAMEWORK
File name: INSTALLUTIL.EXE
File version: *

Publisher: O=MICROSOFT CORPORATION, L=REDMOND,
S=WASHINGTON, C=US
Product name: MICROSOFT® .NET FRAMEWORK
File name: CSC.EXE
File version: *

Publisher: O=MICROSOFT CORPORATION, L=REDMOND,
S=WASHINGTON, C=US
Product name: MICROSOFT® WINDOWS® OPERATING SYSTEM
File name: SC.EXE
File version: *

3.3. Code Execution with DLLs

3.3.1. Demonstrating Attacks

There are several built-in Microsoft programs used to execute DLLs. DLLs can be full-blown malicious applications. However, Microsoft warns that using AppLocker to restrict unwanted DLLs can impact user performance (Microsoft, 2018, August 26). If whitelisting is not used to protect against malicious DLLs, traditional built-in Microsoft programs that execute DLLs should be restricted. A sample Microsoft AppLocker DLL performance warning can be seen in Figure 16.

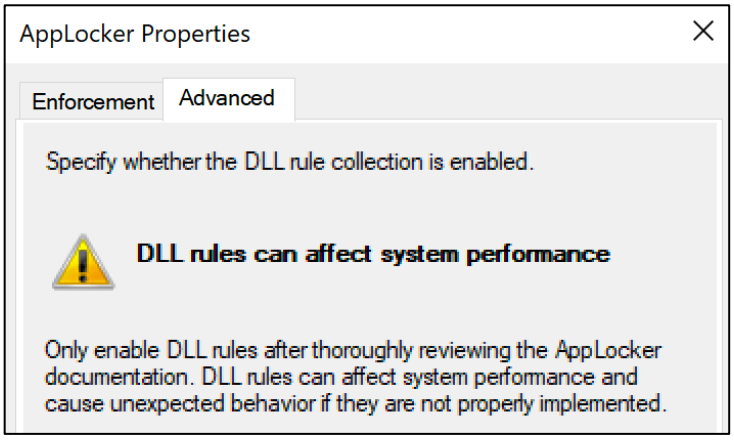


Figure 16. Microsoft's AppLocker notice of potential adverse performance associated with DLL rules.

The attack in this section focuses on the PowerShdll attack library to demonstrate malicious DLLs. On a machine that blocks PowerShell, the code below can give access to a PowerShell command line. To run the attack, download the 64-bit version of powershdll.dll to C:\Exploit on the virtual machine (p3nt4, n.d.).

```
C:\Windows\Microsoft.NET\Framework64\v4.0.30319\regasm.exe /U
        C:\Exploit\PowerShdll.dll
```

If cmd.exe is not available, create a link to regasm.exe and use the above code in the "Target" of the link. If AppLocker's default DLL rules are disabled due to performance concerns, execution of the command succeeds in giving an attacker a PowerShell command prompt. Note that Windows Defender must be disabled for this attack to succeed since Microsoft has previously tagged powershdll.dll as malicious. Successful execution is demonstrated in Figure 17.

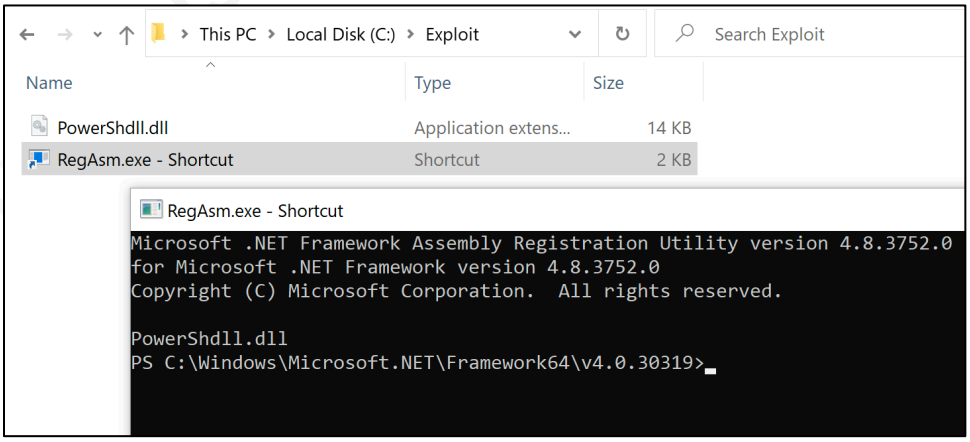


Figure 17. Successful bypass of default AppLocker rules with DLL via regasm.exe if DLL rules are disabled due to performance concerns.

3.3.2. Blocking Attacks with LotL Rules

When LotL rules are enabled, the attack fails. The attack fails even if AppLocker is not being used to block DLLs. AppLocker blocks regasm.exe from executing due to the LotL rules as seen in Figure 18 and Figure 19.

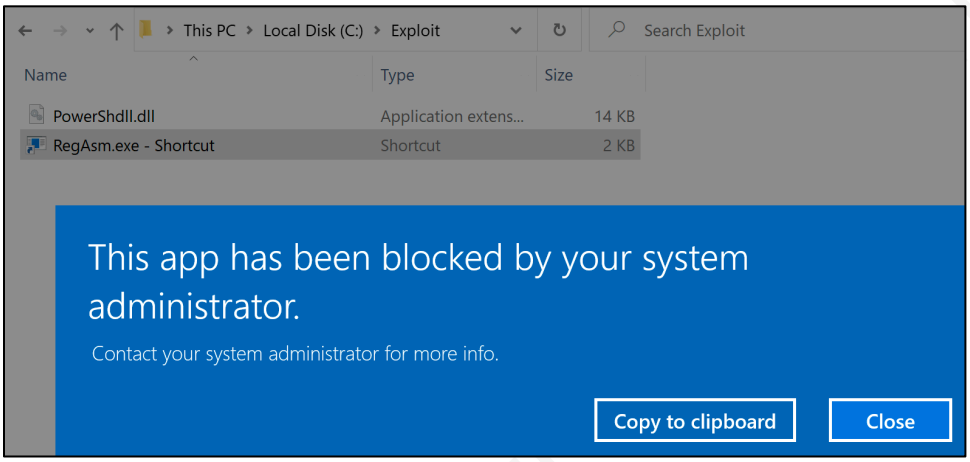


Figure 18. Block of regasm.exe due to AppLocker LotL rules.

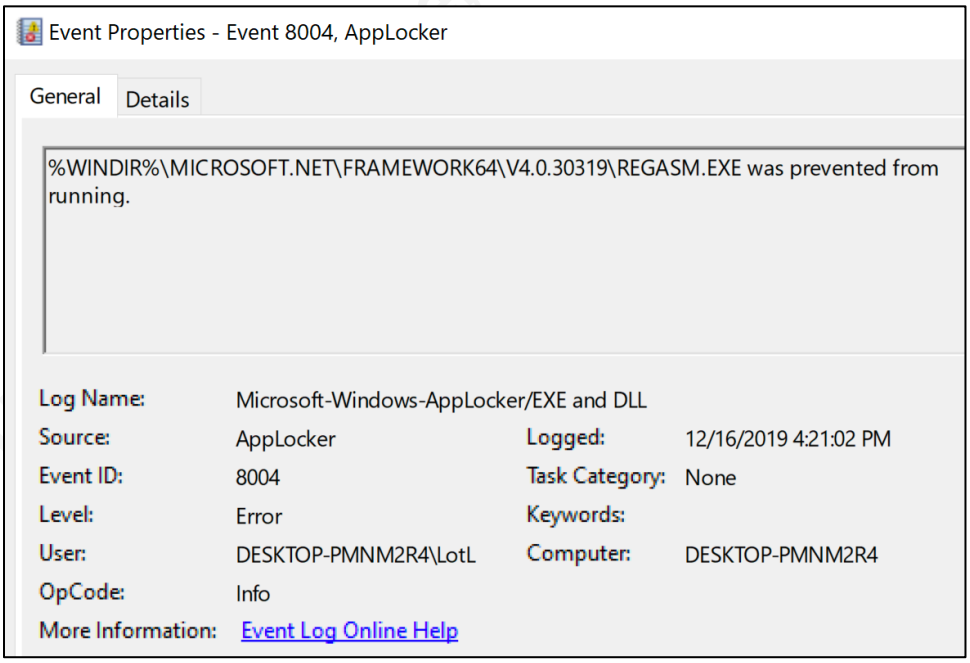


Figure 19. Event Viewer details of regasm.exe block.

For reference, many other built-in Windows programs can be used to abuse DLLs, including rundll32.exe, installutil.exe, regsvcs.exe, regsvr32.exe, msiexec.exe, mavinject.exe, and odbcconf.exe, among others (MITRE, n.d.). Ideally, if AppLocker is not being used to restrict DLLs, all programs that can abuse DLLs should be restricted by the LotL rules to protect against malicious DLLs.

However, depending on the environment, there can be unique problems associated with restricting some of these DLL-executing programs. For example, the Windows

operating system may call rundll32.exe in the context of the current user when a user logs into Windows. If rundll32.exe were to be restricted by LotL rules, any DLLs that the Windows system calls in the context of the current user would be blocked, which could potentially lead to degraded user experience. Closely monitoring AppLocker blocks in the Windows Event Viewer alerts the defender to any unintended blocks. If there are unintended blocks, the defender needs to choose between restricting the offending Windows program, rundll32.exe in this example, or enabling whitelisting restrictions to block all unknown DLLs. One of these two options should be chosen, or an attacker can easily bypass the application whitelisting deployment by utilizing malicious DLLs as demonstrated in the real-world attacks seen in Section 3.3.4 and by Casey Smith at ShmooCon 2015 (Smith, 2015).

Ideally, AppLocker's default DLL rules are enabled along with LotL rules that restrict the majority, if not all, of the DLL-executing programs. The two combined provide a much greater attack surface reduction.

3.3.3. Summary of Behavior

The following chart (Table 3) summarizes how the attack demonstrations behaved for each user. For reference, the Regular user does not have AppLocker enabled. The Default user has the default AppLocker rules enabled and a listing of these rules is located in Appendix C (AppLocker Default Rules). The LotL user has the custom AppLocker LotL rules enabled and a full listing of these rules is provided in Appendix B (AppLocker LotL Rules).

Demonstration of Attack	Regular User	Default User	LotL User
powershdll.dll via regasm.exe	Executed	Blocked DLL file	Blocked regasm.exe
powershdll.dll via regasm.exe without AppLocker DLL rules	Executed	Executed	Blocked regasm.exe

Table 3

See Section 3.3.5 for the implementation of the specific LotL rules that relate to the above attacks.

3.3.4. Real-World Examples

Locky ransomware is a real-world attack where attackers used rundll32.exe to execute the malicious ransomware DLL. This attack distributes malicious JavaScript to victims through an email attachment. When a victim opens the email attachment, wscript.exe executes the JavaScript file, which launches rundll32.exe. Rundll32.exe then runs the malicious DLL (Lawrence, 2016). AppLocker LotL rules block wscript.exe from being executed, which causes the attack to fail when the user opens the JavaScript attachment. However, if the attacker had access to the machine or another means of executing rundll32.exe, the wscript.exe step could be skipped. This attack would then be blocked by the LotL rules when rundll32.exe executes.

Another real-world attack is the PureLocker ransomware attack. This ransomware attack disguises itself as a DLL cryptography library called Crypto++ and launches when an attacker executes regsrv32.exe to run a malicious DLL file (Kajiloti, 2019). With AppLocker LotL rules enabled, regsrv32.exe is blocked, and the attack fails.

3.3.5. Specific LotL rules to Prevent the Attacks

To implement the specific AppLocker LotL rules that prevent the above attacks, create the following AppLocker Executable rules set to Deny for "Employees":

Publisher: O=MICROSOFT CORPORATION, L=REDMOND,
S=WASHINGTON, C=US
Product name: MICROSOFT® WINDOWS® OPERATING SYSTEM
File name: POWERSHELL.EXE
File version: *

Publisher: O=MICROSOFT CORPORATION, L=REDMOND,
S=WASHINGTON, C=US
Product name: MICROSOFT® WINDOWS® OPERATING SYSTEM
File name: POWERSHELL_ISE.EXE
File version: *

Publisher: O=MICROSOFT CORPORATION, L=REDMOND,
S=WASHINGTON, C=US
Product name: MICROSOFT® WINDOWS® OPERATING SYSTEM
File name: RUNDLL32.EXE
File version: *

Publisher: O=MICROSOFT CORPORATION, L=REDMOND,

S=WASHINGTON, C=US
Product name: MICROSOFT® WINDOWS® OPERATING SYSTEM
File name: REGSRV32.EXE
File version: *

Publisher: O=MICROSOFT CORPORATION, L=REDMOND,
S=WASHINGTON, C=US
Product name: MICROSOFT® .NET FRAMEWORK
File name: REGASM.EXE
File version: *

4. Recommendations and Implications

The AppLocker rules referred to in this paper as LotL rules were successful at preventing the living off the land attacks explored in this research. The LotL rules in Appendix B are designed to prevent a wide variety of living off the land attacks, not limited to the living off the land attacks explored in this paper. As seen in the MITRE ATT&K Framework and the LOBAS Project, over 100 Microsoft programs have the potential to be abused by attackers (MITRE, n.d.) (LOBAS, n.d.). Designing strong LotL rules comes down to understanding which Microsoft programs are abused by attackers and then creating specific rules to restrict regular users from executing those programs.

One key to designing LotL rules is to apply them to a user security group and not to "Everyone." In this paper, that security group was called "Employees." This allows the Windows operating system and administrators to have access to the built-in Windows programs while preventing regular users from executing the abused programs. One implication of this design is that administrative users have full access to the abused Microsoft programs. If an attacker gains control of an administrative account, the LotL rules cease to be effective.

4.1. Understanding the Kill-Chain

As explored in the Findings and Discussion section of the paper, certain built-in Windows programs that attackers abuse, such as rundll32.exe, may have a legitimate use for regular users. This makes the decision to restrict specific built-in Windows programs more complicated. Understanding various kill chains for living off the land attacks becomes essential in developing effective LotL rules. If specific programs, such as

rundll32.exe, are not restricted, AppLocker should be used to block unknown DLLs, or LotL rules should be designed to thwart known attacks at other points in the kill chain.

It is important to continually update LotL rules as attackers begin to abuse additional Windows programs. Defenders should create new rules to disrupt the attacks as new kill chains are understood. It is also important to note that AppLocker is only available on Windows Enterprise edition.

4.2. Alerting on Attacks

Setting up alerts on the LotL rules lets defenders know of attacks as they are disrupted by AppLocker so that they can investigate for further signs of malicious activity. Alerts also help identify rules that may need to be turned off in a given environment, such as rundll32.exe.

When initially rolling out the LotL rules, set AppLocker to Audit mode. Watch AppLocker event codes 8003 and 8006. These codes show which applications AppLocker would have blocked if AppLocker were set to Enforce mode. Once LotL rules have been fine-tuned not to disrupt normal system usage, set AppLocker to Enforce mode and begin watching event codes 8004 and 8007. These codes show which applications AppLocker is actively blocking (Microsoft, 2017, September 20).

4.3. Non-Built-In Microsoft Programs

Microsoft programs that are not built-into Windows by default are another essential consideration in designing a whitelisting solution that prevents living off the land attacks. Some whitelisting solutions rely on trusting all program signed by Microsoft. When all Microsoft certificates are trusted, application whitelisting rules designed to prevent living off the land attacks must also block other, non-default Microsoft programs that are abused, such as psexec.exe and bginfo.exe. If these tools aren't already present on the target machine, cybercriminals may download them to the target machine to perpetuate an attack (Palo Alto Networks, n.d.).

5. Conclusion

Living off the land attacks are on the rise (Symantec, 2017). These attacks can be difficult for antivirus vendors to detect since they utilize legitimate built-in Windows programs. Fortunately, AppLocker can be used to detect and thwart living off the land attacks. As seen in this paper, user-based AppLocker rules that restrict abused Windows programs can effectively disrupt these attacks. There should be a significant reduction in successful living off the land attacks as the industry adopts user-based whitelisting rules explored in this research and as the industry continues to migrate towards next generational, behavior-based security tools, such as Windows Defender ATP.

David Brown, mrdavebrown@gmail.com

References

Australian Cyber Security Center (2019). *Implementing Application Whitelisting.* Retrieved from https://www.cyber.gov.au/sites/default/files/2019-12/PROTECT%20-%20Implementing%20Application%20Whitelisting%20%28April%202019%29.pdf

Carbon Black (2019, June 4). *How Carbon Black is Prioritizing Living Off the Land Attacks Part 1.* Retrieved from https://www.carbonblack.com/2019/06/04/how-carbon-black-is-prioritizing-living-off-the-land-attacks-part-1/

CrowdStrike (2019). *2019 Global Threat Report.* Retrieved from https://www.crowdstrike.com/resources/reports/2019-crowdstrike-global-threat-report/

CrowdStrike (2019, May 7). *Going Beyond Malware: The Rise of "Living off the Land" Attacks.* Retrieved from https://www.crowdstrike.com/blog/going-beyond-malware-the-rise-of-living-off-the-land-attacks/

Cybereason Nocturnus (2019, August 5). *Sodinokibi: The Crown Prince of Ransomware.* Retrieved from https://www.cybereason.com/blog/the-sodinokibi-ransomware-attack

Fehrman, B (2016, August 21). *Powershell Without Powershell - How To Bypass Application Whitelisting, Environment Restrictions & AV.* Retrieved from https://www.blackhillsinfosec.com/powershell-without-powershell-how-to-bypass-application-whitelisting-environment-restrictions-av/

David Brown, mrdavebrown@gmail.com

Graham, J (2018, October 26). *Pentesting and .hta (bypassing PowerShell Constrained Language Mode).* Retrieved from https://medium.com/tsscyber/pentesting-and-hta-bypassing-powershell-constrained-language-mode-53a42856c997

Lawrence, A (2016, August 26). *Locky / Zepto Ransomware now being installed from a DLL.* Retrieved from https://www.bleepingcomputer.com/news/security/locky-zepto-ransomware-now-being-installed-from-a-dll/

LOLBAS (n.d.). Living Off The Land Binaries and Scripts (and also Libraries). Retrieved January 9, 2020, from https://lolbas-project.github.io/#

Kajiloti, M (2019, December 11). *PureLocker: New Ransomware-as-a-Service Being Used in Targeted Attacks Against Servers.* Retrieved from https://www.intezer.com/blog-purelocker-ransomware-being-used-in-targeted-attacks-against-servers/

Microsoft (2017, September 20). *Using Event Viewer with AppLocker.* Retrieved from https://docs.microsoft.com/en-us/windows/security/threat-protection/windows-defender-application-control/applocker/using-event-viewer-with-applocker

Microsoft (2018, August 26). *Working with AppLocker rules.* Retrieved from https://docs.microsoft.com/en-us/windows/security/threat-protection/windows-defender-application-control/applocker/working-with-applocker-rules

Microsoft (2019, January 1). *Application Control.* Retrieved from https://docs.microsoft.com/en-us/windows/security/threat-protection/windows-defender-application-control/windows-defender-application-control

MITRE (n.d.). Enterprise Techniques. Retrieved January 9, 2020, from https://attack.mitre.org/techniques/enterprise/

David Brown, mrdavebrown@gmail.com

p3nt4 (n.d.). PowerShdll. Retrieved January 9, 2020, from https://github.com/p3nt4/PowerShdll

Palo Alto Networks (n.d.). *What Are Fileless Malware Attacks and "Living Off the Land"? Unit 42 Explains.* Retrieved January 9, 2020, from https://www.paloaltonetworks.com/cyberpedia/what-are-fileless-malware-attacks

PowerShellMafia (n.d.). PowerSploit. Retrieved January 9, 2020, from https://github.com/PowerShellMafia/PowerSploit/blob/master/Privesc/PowerUp.ps1

PwC (2017). *Operation Cloud Hopper.* Retrieved from https://www.pwc.co.uk/cyber-security/pdf/cloud-hopper-annex-b-final.pdf

Rapid7 (2019, November 19). *Quarterly Threat Report: Q3 2019.* Retrieved from https://www.rapid7.com/research/report/2019-q3-threat-report/

Smith, C (2015). *Simple Windows Application Whitelisting Evasion* [Video file]. Retrieved from https://www.youtube.com/watch?v=XVuboBH5TYo

Symantec (2017). *Living off the land and fileless attack techniques.* Retrieved from https://www.symantec.com/content/dam/symantec/docs/security-center/white-papers/istr-living-off-the-land-and-fileless-attack-techniques-en.pdf

Trend Micro (2016, June 16). *New RAA Ransomware Uses Only JavaScript to Infect Computers.* Retrieved from https://www.trendmicro.com/vinfo/nl/security/news/cybercrime-and-digital-threats/new-raa-ransomware-uses-only-javascript-to-infect-computers

Vipre (2018, May 25). Sequre Ransomware compiles its own source code. Retrieved from https://labs.vipre.com/sequre-ransomware-compiles-its-own-source-code/

David Brown, mrdavebrown@gmail.com

Appendix A (Virtual Machine Setup)

Windows Enterprise edition is required for AppLocker. To run AppLocker, launch services.msc from the command prompt and set "Application Identity" to Automatic.

Figure 20. Winver.exe for virtual machine used in testing.

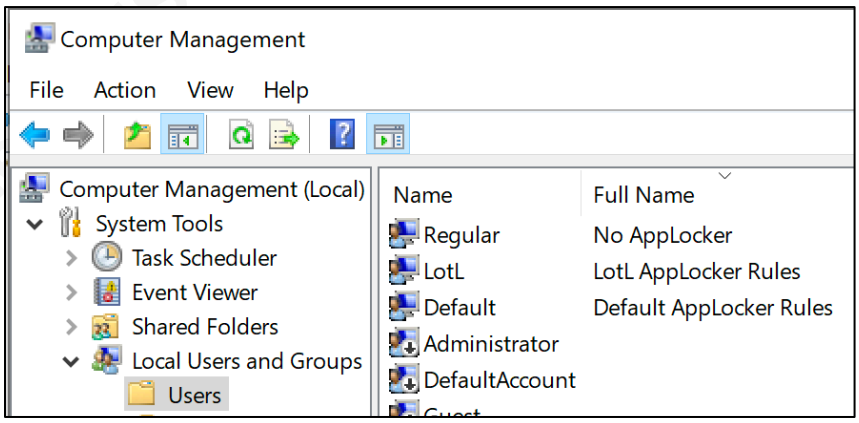


Figure 21. Compmgmt.msc view of users used in testing.

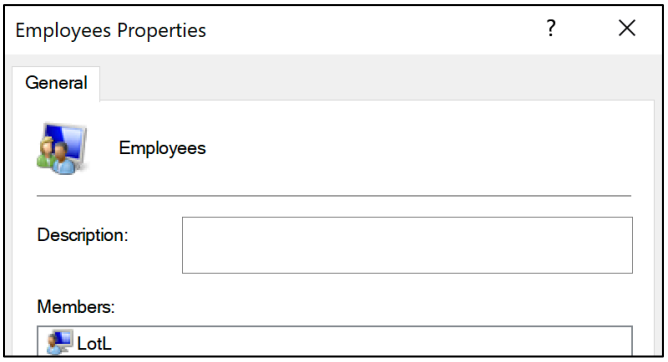


Figure 22. View of Employees group membership used in testing.

David Brown, mrdavebrown@gmail.com

Appendix B (AppLocker LotL Rules)

Executable Rules:

Deny	DESKTOP-PMNM2R4\Employees	ADDINPROCESS.EXE, in MICROSOFT...	Publisher
Deny	DESKTOP-PMNM2R4\Employees	ADDINPROCESS32.EXE, in MICROSO...	Publisher
Deny	DESKTOP-PMNM2R4\Employees	ADDINUTIL.EXE, in MICROSOFT® .N...	Publisher
Deny	DESKTOP-PMNM2R4\Employees	AT.EXE, in MICROSOFT® WINDOWS...	Publisher
Deny	DESKTOP-PMNM2R4\Employees	BCDEDIT.EXE, in MICROSOFT® WIN...	Publisher
Deny	DESKTOP-PMNM2R4\Employees	BITSADMIN.EXE, in MICROSOFT® W...	Publisher
Deny	DESKTOP-PMNM2R4\Employees	CERTUTIL.EXE, in MICROSOFT® WIN...	Publisher
Deny	DESKTOP-PMNM2R4\Employees	CMD.EXE, in MICROSOFT® WINDO...	Publisher
Deny	DESKTOP-PMNM2R4\Employees	CMDKEY.EXE, in MICROSOFT® WIN...	Publisher
Deny	DESKTOP-PMNM2R4\Employees	CMSTP.EXE, in MICROSOFT(R) CON...	Publisher
Deny	DESKTOP-PMNM2R4\Employees	CONTROL.EXE, in MICROSOFT® WI...	Publisher
Deny	DESKTOP-PMNM2R4\Employees	CSC.EXE, in MICROSOFT® .NET FRA...	Publisher
Deny	DESKTOP-PMNM2R4\Employees	CSCRIPT.EXE	Publisher
Deny	DESKTOP-PMNM2R4\Employees	ESENTUTL.EXE, in MICROSOFT® WI...	Publisher
Deny	DESKTOP-PMNM2R4\Employees	EVENTVWR.EXE, in MICROSOFT® W...	Publisher
Deny	DESKTOP-PMNM2R4\Employees	EXTEXPORT.EXE, in INTERNET EXPLO...	Publisher
Deny	DESKTOP-PMNM2R4\Employees	GPRSLT.EXE, in MICROSOFT® WIND...	Publisher
Deny	DESKTOP-PMNM2R4\Employees	INFDEFAULTINSTALL.EXE, in MICROS...	Publisher
Deny	DESKTOP-PMNM2R4\Employees	INSTALLUTIL.EXE	Publisher
Deny	DESKTOP-PMNM2R4\Employees	MAVINJECT32.EXE, in MICROSOFT® ...	Publisher
Deny	DESKTOP-PMNM2R4\Employees	MAVINJECT64.EXE, in MICROSOFT® ...	Publisher
Deny	DESKTOP-PMNM2R4\Employees	MICROSOFT.WORKFLOW.COMPILER...	Publisher
Deny	DESKTOP-PMNM2R4\Employees	MMC.EXE, in MICROSOFT® WINDO...	Publisher
Deny	DESKTOP-PMNM2R4\Employees	MSBUILD.EXE	Publisher
Deny	DESKTOP-PMNM2R4\Employees	MSDT.EXE	Publisher
Deny	DESKTOP-PMNM2R4\Employees	MSHTA.EXE	Publisher
Deny	DESKTOP-PMNM2R4\Employees	MSIEXEC.EXE, in WINDOWS INSTALL...	Publisher
Deny	DESKTOP-PMNM2R4\Employees	MSINFO.DLL, in MICROSOFT® WIN...	Publisher
Deny	DESKTOP-PMNM2R4\Employees	NET.EXE, in MICROSOFT® WINDO...	Publisher
Deny	DESKTOP-PMNM2R4\Employees	ODBCCONF.EXE, in MICROSOFT® ...	Publisher
Deny	DESKTOP-PMNM2R4\Employees	POWERSHELL.EXE, in MICROSOFT® ...	Publisher
Deny	DESKTOP-PMNM2R4\Employees	POWERSHELL_ISE.EXE, in MICROSOF...	Publisher
Deny	DESKTOP-PMNM2R4\Employees	PRESENTATIONHOST.EXE, in MICRO...	Publisher
Deny	DESKTOP-PMNM2R4\Employees	REGASM.EXE	Publisher
Deny	DESKTOP-PMNM2R4\Employees	REGEDIT.EXE, in MICROSOFT® WIN...	Publisher

Deny	DESKTOP-PMNM2R4\Employees	REGSVCS.EXE	Publisher
Deny	DESKTOP-PMNM2R4\Employees	REGSVR32.EXE	Publisher
Deny	DESKTOP-PMNM2R4\Employees	RUNDLL32.EXE, in MICROSOFT® WI...	Publisher
Deny	DESKTOP-PMNM2R4\Employees	SC.EXE, in MICROSOFT® WINDOWS...	Publisher
Deny	DESKTOP-PMNM2R4\Employees	SCTASKS.EXE, in MICROSOFT® WIN...	Publisher
Deny	DESKTOP-PMNM2R4\Employees	VSSADMIN.EXE, in MICROSOFT® WI...	Publisher
Deny	DESKTOP-PMNM2R4\Employees	WEVTUTIL.EXE, in MICROSOFT® WI...	Publisher
Deny	DESKTOP-PMNM2R4\Employees	WMIC.EXE, in MICROSOFT® WIND...	Publisher
Deny	DESKTOP-PMNM2R4\Employees	WSCRIPT.EXE, in MICROSOFT ® WI...	Publisher

Figure 23. AppLocker Executable LotL rules used in testing.

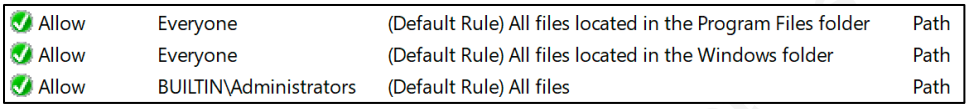
DLL Rules:

Deny	DESKTOP-PMNM2R4\Employees	ADVPACK.DLL, in INTERNET EXPLORER, fr...	Publisher
Deny	DESKTOP-PMNM2R4\Employees	DFSHIM.DLL, in MICROSOFT® WINDOW...	Publisher
Deny	DESKTOP-PMNM2R4\Employees	SETUPAPI.DLL, in MICROSOFT® WINDO...	Publisher
Deny	DESKTOP-PMNM2R4\Employees	SYSSETUP.DLL, in MICROSOFT® WINDO...	Publisher
Deny	DESKTOP-PMNM2R4\Employees	SYSTEM.MANAGEMENT.AUTOMATION.DLL	Publisher

Figure 24. AppLocker DLL LotL rules used in testing.

Appendix C (AppLocker Default Rules)

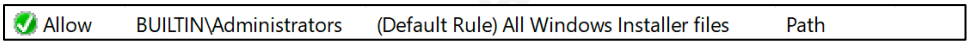
Executable Rules:



Allow	Everyone	(Default Rule) All files located in the Program Files folder	Path
Allow	Everyone	(Default Rule) All files located in the Windows folder	Path
Allow	BUILTIN\Administrators	(Default Rule) All files	Path

Figure 25. Default AppLocker Executable Rules used in testing.

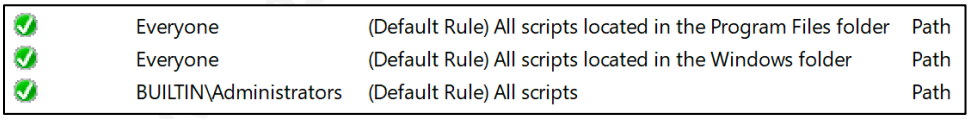
Windows Installer Rules (note – deleted default "Everyone" rules to enhance the security of default rules):



Allow	BUILTIN\Administrators	(Default Rule) All Windows Installer files	Path

Figure 26. Default AppLocker Installer Rules used in testing.

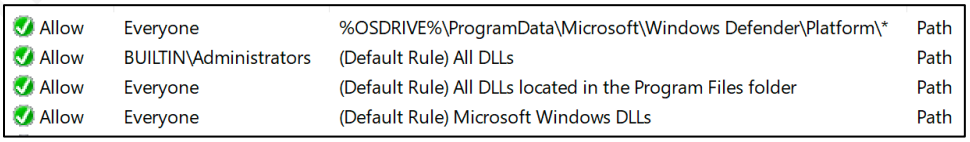
Script Rules:



	Everyone	(Default Rule) All scripts located in the Program Files folder	Path
	Everyone	(Default Rule) All scripts located in the Windows folder	Path
	BUILTIN\Administrators	(Default Rule) All scripts	Path

Figure 27. Default AppLocker Script Rules used in testing.

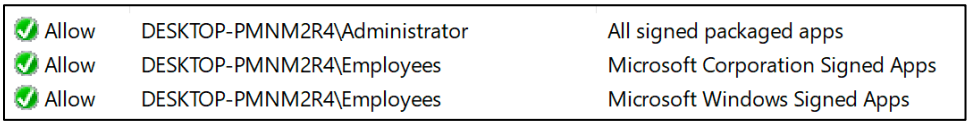
DLL Rules (note – added Windows Defender path to allow Defender to run properly):



Allow	Everyone	%OSDRIVE%\ProgramData\Microsoft\Windows Defender\Platform*	Path
Allow	BUILTIN\Administrators	(Default Rule) All DLLs	Path
Allow	Everyone	(Default Rule) All DLLs located in the Program Files folder	Path
Allow	Everyone	(Default Rule) Microsoft Windows DLLs	Path

Figure 28. Default AppLocker DLL Rules used in testing.

Package app Rules (note – removed default "Everyone" rule and added Microsoft signed rules to enhance the security of default rules):



Allow	DESKTOP-PMNM2R4\Administrator	All signed packaged apps
Allow	DESKTOP-PMNM2R4\Employees	Microsoft Corporation Signed Apps
Allow	DESKTOP-PMNM2R4\Employees	Microsoft Windows Signed Apps

Figure 29. Default AppLocker Package App Rules used in testing.

David Brown, mrdavebrown@gmail.com

Appendix D (program.cs)

```
using System;
using System.Configuration.Install;
using System.Runtime.InteropServices;
using System.Management.Automation.Runspaces;
public class Program {
 public static void Main() {}
}
[System.ComponentModel.RunInstaller(true)]
public class Sample: System.Configuration.Install.Installer {
 public override void Uninstall(System.Collections.IDictionary savedState) {
  Mycode.Exec();
 }
}
public class Mycode {
 public static void Exec() {
  string command = System.IO.File.ReadAllText(@ "C:\Exploit\PowerUp.ps1");
  RunspaceConfiguration rspacecfg = RunspaceConfiguration.Create();
  Runspace rspace = RunspaceFactory.CreateRunspace(rspacecfg);
  rspace.Open();
  Pipeline pipeline = rspace.CreatePipeline();
  pipeline.Commands.AddScript(command);
  pipeline.Invoke();
 }
}
```

Figure 30. Program.cs source code (Fehrman, 2016, August 21).