

TALLINN UNIVERSITY OF TECHNOLOGY

School of Information Technologies

Holger Rünkaru 204781IVCM

**METHOD FOR DEFENDING AGAINST LIVING OFF THE
LAND ATTACKS WITH BUILT-IN FEATURES OF
WINDOWS**

Master's Thesis

Academic Supervisor

Pavel Tšikul

PhD Candidate

Tallinn 2022

Author's declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Holger Rünkaru

.....

(signature)

Date: May 16, 2022

Annotatsioon

Käesoleva lõputöö eesmärgiks on uurida usaldusväärseid komponente kuritarvitavaid ründeid *Windows* operatsioonisüsteemile ning analüüsida *Windows* operatsioonisüsteemi osisteks olevate kaitsefunktsioonide võimakusi, piiranguid ning eripärasid, et luua laiaulatuslik kaitse rünnete vastu.

Analüüs, poliitikate loomine ning rakendamine on võimalusel teostatud läbi skriptimise, et tõestada kontseptsioonina automatiseerimise võimalikus ning kujutada automatiseerimise protsessi. Töö kirjeldab *Windows Defender Application Control*, *AppLocker* ning *Access Control List* kaitsefunktsionaalsuste omadusi, disaini põhimõtteid, ning viise kuidas kaitsefunktsioone koos kasutades saavutada paindlikum kaitse.

Lõputöö tulemuseks on kokkuvõtlik sammude loend, mida läbida kaitsefunktsioonide juurutamisel, viidates varasematele töö osadele detailsete kirjeldustega, mille abil on võimalik luua ühtne ja lihtsustav kasutatud turvafunktsioonide haldus- ning dokumenteerimislahendus.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 93 leheküljel, 5 peatükki, 15 joonist, 2 tabelit.

Abstract

This thesis aims to study attacks abusing trusted components of the Windows operating system and analyses the capabilities, limitations, and essential considerations of the Windows operating system's built-in security features to create a comprehensive defense.

The analysis, implementation, and deployment were performed by writing scripts, where applicable, to perform as a proof-of-concept for the automation approach and to illustrate the process. The work covers features, design principles, and implementation methods of the Windows security features Windows Defender Application Control, AppLocker, and Access Control List.

The outcome of this thesis is a comprehensive sequence of steps with references to sections providing details for building organization-specific integrated and simplified management and documentation solution for the security features.

The thesis is in English and contains 93 pages of text, 5 chapters, 15 figures, 2 tables.

List of abbreviations and terms

ACE	Access Control Entry
ACL	Access Control List
CSIRT	Cyber Security Incident Response Team
CSV	Comma-Separated Values
DACL	Discretionary Access Control List
GPO	Group Policy Object
GUI	Graphical User Interface
IOC	Indicator of compromise
LOLBAS project	Living Off The Land Binaries, Scripts and Libraries project
LotL	Living off the Land
SACL	System Access Control List
SID	Security Identifier
WDAC	Windows Defender Application Control

Table of Contents

List of Figures	8
List of Tables	9
1 Introduction	10
1.1 Motivation	10
1.2 Objectives	11
1.3 Thesis outline	11
1.4 Acknowledgements	12
2 Background	13
2.1 Living off the Land attacks	13
2.1.1 The popularity the LotL technique	13
2.1.2 Types of LotL attacks	13
2.1.3 Living Off The Land Binaries, Scripts and Libraries project	14
3 Related research	16
4 Methodology	19
4.1 General description	19
4.2 Limitations of the study	19
4.2.1 Suitable target organizations	20
4.3 Lab setup	21
4.3.1 Workstation profile	22
4.4 Identifying points of contact	22
4.4.1 Parsing filepaths from LOLBAS project	23
4.4.2 Finding dual-use files existing in default Windows installation	23
4.4.3 Other files in the system with same signer	24
4.5 Application Control for Windows	25
4.5.1 Cost of Application Control	26
4.5.2 Windows Defender Application Control	27
4.5.3 AppLocker	30
4.5.4 AppLocker rules	31
4.6 Access Control List	32
4.6.1 Example of manage-bde.wsf	33
4.6.2 Design of DACL rules	34

4.7	Protecting against attacks via Powershell	35
4.7.1	Powershell Constrained Language mode	36
4.8	Implementing the rules	37
4.8.1	Layers of protection on high-level	37
4.8.2	Implementing Windows Defender Application Control	38
4.8.3	Implementing AppLocker	42
4.8.4	Implementing Access Control List	47
4.9	Testing the rules	48
4.9.1	Testing WDAC policies	49
4.9.2	Testing AppLocker policies	50
4.9.3	Testing the DACL for manage-bde.wsf	54
4.10	Results of the study	55
4.10.1	Limitations of the defense	56
4.10.2	Meeting the set objectives	56
4.10.3	Foundations for building the semi-automated process	57
4.10.4	Additional/optional features	59
5	Summary	61
5.1	Future work	62
5.1.1	Development of a management tool with user-friendly interface based on the method proposed	62
5.1.2	Evaluating the described Living off the Land defenses in enterprise environment	62
5.1.3	Analysis of recommended drivers block rules by Microsoft	62
5.1.4	Using Just Enough Administration for locking down Powershell in Workstations	63
	Bibliography	64
	Appendices	72
	Appendix 1 - LOLBAS-filepaths.py	72
	Appendix 2 - Get-LOTLInfo.ps1	76
	Appendix 3 - LOLBAS_WDAC-policy.py	78
	Appendix 4 - AppLocker-example.xml	81
	Appendix 5 - LOLBAS_AppLocker-policy.py	85

List of Figures

1	<i>Application Control cost comparison [18]</i>	27
2	<i>Calc.exe execution from manage-bde.wsf</i>	34
3	<i>DACLs and ACEs [48]</i>	35
4	<i>Deploy Code Integrity Policy</i>	41
5	<i>Event Viewer - 7010 - Enabled Code Integrity Policy</i>	41
6	<i>AppLocker UI - Export and Import</i>	44
7	<i>Group Policy Management Editor - AppLocker Implemented</i>	45
8	<i>AppLocker Properties - Enable AppLocker</i>	45
9	<i>AppLocker Properties - Enforce AppLocker</i>	46
10	<i>Group Policy Management Editor - File System</i>	47
11	<i>DACL - manage-bde.wsf</i>	48
12	<i>Event Viewer - WDAC Event ID 3077</i>	50
13	<i>Event Viewer - AppLocker ID 8003, calc.exe</i>	53
14	<i>Get-AuthenticodeSignature, calc2.exe</i>	54
15	<i>DACL tests - manage-bde.wsf</i>	55

List of Tables

1	<i>LOLBAS entity path count distribution by types.</i>	23
2	<i>LOLBAS files existence and signature</i>	24

1. Introduction

1.1 Motivation

The malware detection capabilities have come a long way compared to the early days with the introduction of behavior monitoring, machine learning detections, and real-time Indicator-of-compromise (IOC) sharing capabilities; this has nudged the offensive side to move towards alternative approaches to bypass the security solutions [1].

The use of novel defense evasion methods by the attackers and the defenders implementing new prevention and detection capabilities is a constant cat and mouse game between the two parties.

It is easy for IT teams and cybersecurity professionals to take the mental shortcut to deem binaries signed by Microsoft as secure components of the Windows operating system and discard the related logs to reduce noise. Even default configurations for Windows operating system security features often allowlist binaries considered part of operating system[2] - perhaps understandable from the perspective of Microsoft as the vendor, the product needs to "just work" out-of-the-box.

Such defaults are not necessarily secure, being the primary motivator for the study. Without the protection against Living off the Land techniques, the attackers have numerous ways for defense evasion, including evading allowlisting itself and providing access to built-in utilities to achieve execution, privilege escalation, credential access, and other tactics[3]. The defense against bypassing the security measure of allowlisting itself seems to be often missing from existing papers describing the tools.

The Living off the Land technique – meaning the attackers are not introducing additional binaries to victim systems but utilizing the trusted resources built-in to the operating system or using already installed trusted software in the victim system [1]. Many of these tools are used by system administrators for legitimate work, making it even harder to detect or completely block [4]. More information on the Living off the Land technique is in Chapter 2.

The primary goal of this paper is to provide a methodology on how to use existing intelligence of Living off the Land techniques and binaries and how to use the built-in security features of the Windows operating system to defend against the attacks. The approach should be easily manageable, and it should simplify the work processes of administrators responsible for deployment by utilizing automation. The procedure should also allow using configurations to track the changes and reasoning to enable the tool to be incorporated into the policy management and decision-making process.

The secondary goal of this paper is to bring more focus to the potential pitfalls in the default configurations for Windows application control mechanisms and how it can lead to bypassing the application control security measures entirely.

1.2 Objectives

The overall aim is to provide documented analysis and a method to reduce the success and impact of attacks on Windows operating systems by reducing the attack surface for Living off the Land techniques, focusing on dual-use binaries in the system. The task needs to be split into smaller objectives to achieve the goal.

The first objective is to gather background information on the Living off the Land attack types, collect information on specific techniques, and compare the data with information captured from the Windows operating system to identify contact points.

The second objective is to analyze the capabilities, design principles, and limitations of the built-in Windows security measures available for attack-surface reduction from Living off the Land attacks to provide the approach on how to layer the measures for comprehensive defense, including technical information on implementation procedures.

The third objective is to utilize the knowledge from the first two objectives to generate a methodology on how to build a solution to reach the set goal of defending against Living off the Land attacks. This objective includes using a programmatic approach during the research and publishing all the scripts used for information gathering, configuration creation, and policy deployment as proof of concept.

1.3 Thesis outline

Chapter 1 states the motivation for the study and provides objectives for the work. Chapter 2 adds background information in regards to the paper. In Chapter 3, related works are

presented. Chapter 4 starts with a general description, limitations, and suitable target organizations and then dives into the design, implementation, testing, and automating of the solution for the problem. Chapter 5 covers the summary and thoughts for future works. At the end of the paper, the Appendixes provide example configuration and scripts used in the study.

1.4 Acknowledgements

I want to express my gratitude to my colleague, experienced Microsoft expert, Meelis Nigols, for his invaluable perspective on Windows internals with a heavy focus on security and always finding time for insightful technical discussions.

2. Background

2.1 Living off the Land attacks

2.1.1 The popularity the LotL technique

According to the IBM X-Force team, 57% of the encountered attacks did not include malicious files and were based on Living off the Land techniques [1]. Symantec is reporting continuing growth of attacks relying solely on living off the land techniques and not utilizing any malicious code [4].

In 2019 Q3 Quarterly Threat Report by Rapid7 one of three key takeaways is that attackers are using what's available on systems to continue their campaigns, referring to Living off the Land attacks[5].

2.1.2 Types of LotL attacks

When looking into Living off the Land attacks, one is likely to stumble on the closely related term 'fileless attack' or 'fileless malware', which may be used almost interchangeably with Living off the Land attacks. There are different classifications for the fileless approach [4][6] showing that only part of the approaches is truly fileless, and in other cases, data might be written to disk or existing files used.

Type 1 [6] or Memory-only [4] – truly fileless attacks. Typically get the code execution by taking advantage of some security vulnerability and injecting the code into memory. Most are seen in self-replicating worm-type malware like SQL slammer [7]. The infection stays in the memory, but since it does not modify system files, it also does not have persistence – but in case there are enough vulnerable machines and no mitigations in place, the machine would likely get re-infected after rebooting. In some cases, the objectives of attacks could also be achieved within a short time period, and persistence is not a must, with the benefit of leaving fewer traces for analysts.

Type 2 [6] or Fileless persistence [4] and non-PE files [4] – files are used indirectly. In this stage, the malicious code is usually already running on the machine, but to achieve

persistence, data must be written to the system, for example, into Registry, Group Policy Objects, Scheduled tasks, or existing files. The data is not written to a separate file, but it still resides on the disk. The data can be a shellcode or utilize scripting frameworks like Powershell, WScript, CScript, and Javascript.

In some cases, a hybrid approach can be used – when booting the system up, the malware loads into memory, and the persistence mechanism is removed from the disk – when shutting the machine down, the malware writes it back to the system data on disk to be loaded on next start-up.

Type 3 [6] or Dual-use tools [4]– files needed to operate. One method is to take advantage of random files with specific extensions by configuring the system to execute payload code when such files are opened, for example, as debugger [8]. The file execution could be a common extension, like .txt or .jpeg, and the attacker is hoping that the end-user will open some file with a suitable extension in a reasonable timeframe, or a suitable file could be configured to be opened automatically on the start-up of the system.

This group also includes Dual-use tools which consist of system tools and legitimate applications.

2.1.3 Living Off The Land Binaries, Scripts and Libraries project

The open-source project of 'Living Off The Land Binaries, Scripts and Libraries' (LOLBAS) started as a list of living off the land binaries and living off the land scripts in the Windows operating system, LOLBins, and LOLScripts, respectively, later libraries were added.

The project was started in 2018 by Oddvar Moe, and at the time of writing, the LOLBAS project has six listed maintainers, 59 contributors, and 657 forks in GitHub [9]. As the term is widely used by well-known security researchers and companies it can be considered de-facto standard [1][4][5][10][11].

A suitable binary, script, or library for the project must meet the following criteria [9]:

- Be a Microsoft-signed file, either native to the OS or downloaded from Microsoft
- Have extra "unexpected" functionality. It is not interesting to document intended use cases.
 - Exceptions are application allowlisting bypasses
- Have functionality that would be useful to an APT or red team

The LOLBAS project is a valuable resource for both the offensive and defensive sides to either find ways to bypass security controls and take advantage of the targeted systems' trusted files or get to know where one should focus on making sure to be able to detect or prevent the misuse.

3. Related research

The relevance of the Living off the Land attacks is evident from both academic and non-academic coverage.

In the 2021 IEEE Symposium on Security and Privacy, analysis was performed across several malware datasets, containing a total of 31,805,549 samples - the average prevalence of the Living off the Land technique was found to be 9.41% - however, the prevalence in Advanced Persistent Threat malware reached 26.26% [1]. The same analysis also found that there is a generalized detection gap in 10 of the most popular anti-virus products when testing Living off the Land techniques on fully patched Windows systems[1].

Symantec (acquired by Broadcom in August 2019) published an Internet Security Threat Report (ISTR) in 2017, focusing specifically on the Living off the Land technique and stating it to be a clear trend[4]. In 2019 ISTR publication claimed that the Living off the Land trend shows no sign of abating and a significant increase in certain activities[12]. In the 2022 Symantec paper on Ransomware Threat Landscape, the popularity of the Living off the Land technique is stated as one of the main findings[13].

In the 2022 Global Threat Report by CrowdStrike, it has been said that 62% of attacks observed by CrowdStrike in Q4 of 2021 were malware-free and utilizing Living off the Land techniques[14].

In the 2021 Threat Detection Report by Red Canary, the specific MITRE ATT&CK technique T1218 Signed Binary Process Execution[15] using Living off the Land approach to bypass the allowlisting defenses itself was listed as the top 2 technique with 19% prevalence[16] and is illustrating the need for prevention of this type of attacks.

While looking into academic works on the allowlisting features of the Windows operating system, we have to go further back to find the sources.

In 2017 IEEE published a paper by Rohan Durve and Ahmed Bouridane, where it was demonstrated that application control, in general, is often avoided due to the high cost of maintaining the exhaustive list of applications for allowlisting approach[17]. While second

paper from 2014, used as a reference by R. Durve and A. Bouridane in their work, shows that the cost of incident response and recovery is likely even higher[18].

Although, it was suggested by R. Durve and A. Bouridane that together with Device Guard allowlisting approach with the ability to use rules based on software publishers will significantly lower the costs[17]. The cost can be lowered even more by utilizing the automatically generated policies by scanning a 'golden image' machine - a pre-audited machine to be running only trusted software[17].

The study by R. Durve and A. Bouridane illustrates the importance of application control[17]. It provides valuable information on how to proceed with implementing the approach provided by the author in the work at hand when there are no existing application control policies to improve[17].

The R. Durve and A. Bouridane also noted a limitation in their work - their approach is completely ineffective against threats that exploit software vulnerabilities in trusted applications AND then exist purely within memory[17].

The author would argue that the second qualifier does not have to be true necessarily, proved by the amount of allowlisting bypass opportunities listed in the LOLBAS project[19]. Also, all of the listed binaries in the LOLBAS project[19] are trusted by the Windows operating system, proving the first qualifier of the limitation given earlier - allowlisting without defending against bypassing techniques is ineffective - demonstrating the first research gap to be covered by the author in the paper at hand.

A technical paper by David Brown, published in 2020, demonstrates the relevance and importance of the defense against Living off the Land attacks[10]. The paper covers how AppLocker can be used for blocklisting the binaries used for Living off the Land attacks while stating that the recommendation by Microsoft is to use Windows Defender Application Control[10].

The reasoning for using AppLocker by D. Brown is relevant. However, the author believes that building a comprehensive solution requires utilizing more tools (AppLocker, Windows Defender Application Control, and Access Control List), as shown in the paper. The more flexible approach of utilizing Windows Defender Application Control provides more robust protection in terms of defense evasion resilience while still using AppLocker to further lock down the tools utilized only by specific user groups and using Access Control List to cover the gaps where Windows Defender Application Control and AppLocker are unable to defend. The comprehensive approach brings novelty to the use of built-in allowlisting

tools in Windows.

The second piece of novelty comes from the semi-automated design proposed in the study at hand. It can simplify the overhead of managing multiple solutions and make deployments quicker and more flexible while reducing management costs with automation. Contributing the method as a foundation for developing a dedicated management tool.

4. Methodology

4.1 General description

The research is approached by using experimental methods. The work started with providing necessary background information and continues with setting up expectations for the environment where described defensive measures would be applicable.

Following is an analysis of files existing in default installations of the Windows operating system, which are considered part of the LOLBAS project and considered a potential threat. LOLBAS project was further described in Chapter 2.1.3. Including analysis of signature data for both LOLBAS files and operating system files.

Then applicable Microsoft defensive features included in the Windows operating system are analyzed to create a design and deployment plan. This section covers Windows Defender Application Control, AppLocker, and Access Control Lists.

Testing of the designed rules is next, with an analysis of testing results. Description for automation foundations is provided.

The study is concluded by identifying gaps needing further attention in other publications.

4.2 Limitations of the study

As described in Chapter 2.1.2 there are multiple types of Living off the Land attacks. The primary focus of the study is to find methods to protect operating systems from attacks utilizing Type 3 / Dual-use tools. More specifically, the dual-use tools signed by Microsoft, as such files have the highest chance of bypassing the detection and/or prevention methods.

The files described are mapped by multiple security researchers already in the LOLBAS project covered in Chapter 2.1.3. The criteria set in the LOLBAS project are suitable for the study at hand, and the files listed are the basis for this work.

It is important to note that, although not in the scope of this work, there are many other

legitimate utilities and tools with dual-use features not signed or developed by Microsoft. These tools can have a high prevalence in the tool kits of system administrators and often also be used for malicious purposes.

For example, many NirSoft utilities [20] can be used to extract web browsing history, monitor network traffic, and recover passwords. Nmap tool is a network scanner with various legitimate uses, also widely used in non-legitimate ways.

And then there is Mimikatz [21] - at first developed as a research project when Microsoft disregarded the vulnerability report regarding WDigest by Mimikatz author Benjamin Delpy. Now Mimikatz is used mainly as a hacking tool by many threat actors, including organized crime and state-sponsored groups [22].

4.2.1 Suitable target organizations

Organizations looking into implementing the defensive measures described in the study fully, in allowlisting mode, should be with mature IT processes and resources to successfully handle potential extra load in the initial deployment phase and continuously update the rulesets based on business requirements. The baseline security should be strict, and endpoints should be managed by the organization, including central software management, to take full advantage of the approach.

Some of the most suitable target organizations for allowlisting approach could include vital service providers, national security organizations, financial institutions, and other enterprises with heightened security requirements and willingness to manage software deployed in their IT environment.

For the blocklisting approach only, all organizations with medium cyber security and IT maturity levels could be considered suitable targets.

Based on Microsoft Windows Defender Application Control documentation, the organizations should comply with the following [23]:

- You have deployed or plan to deploy the supported versions of Windows in your organization.
- You need improved control over the access to your organization's applications and the data your users access.
- Your organization has a well-defined process for application management and deployment.

- You have resources to test policies against the organization's requirements.
- You have resources to involve Help Desk or to build a self-help process for end-user application access issues.
- The group's requirements for productivity, manageability, and security can be controlled by restrictive policies.

For best results, the workstation users should not have administrative privileges, with exceptions only for Helpdesk and some IT pro users.

Organizations should also consider other essential security measures, for example, cyber-hygiene training for users, anti-malware solutions, e-mail security, full disk encryption of mobile endpoints, and physical security. These measures are out of the scope of this work and are considered a prerequisite for mature organizations.

The importance of the restrictions described here will become evident in the implementation and automation phases later in work.

4.3 Lab setup

Few virtual machines are deployed for gathering information, deploying designed policies, testing the created scripts, and testing the policies' results.

A Windows Server 2016 Standard is used for management tasks, but the specific version has little effect. A majority of testing and results would be done and seen on the side of the workstation. Windows Server specifics should not affect the Group Policies applied on workstations.

For testing of the policies, two workstation virtual machines are deployed - one as control and the second for validating test results with applied Group Policies to deploy security features. More on these Group Policy settings in the implementation section 4.8.

Both machines would also have the same default Group Policy to modify the following settings:

- enable Remote Desktop Connection
- add firewall rules for Remote Desktop Connections
- add Domain LabUsers group to local Remote Desktop User group
- disable sleep in Power Management
- stop automatic updates

- add domain LabAdmins group to local Administrators group

4.3.1 Workstation profile

Due to using Group Policy for Windows Defender Application Control settings management, it is required to use the Enterprise edition of Windows. Also, the Windows Defender Application Control base policy files are only included in the filesystem with the Enterprise edition.

The specific version of Windows 10 Enterprise operating system used is 21H2, with build 19044.1288, with following hotfixes:

- KB5004331
- KB5003791
- KB5006670
- KB5005699

To ensure a consistent testing environment throughout the process, automatic Microsoft Updates were disabled - the intention of the study is not to provide methods dependent on updates or Windows build. The updates are disabled to ensure consistent results if some tests require re-running.

Updating was disabled by applying the following Group Policy setting:

*Computer Configuration > Administrative Templates > Windows Components > Windows Update > **Configure Automatic Updates - Disabled***

4.4 Identifying points of contact

To be able to analyze the methods to protect Windows 10 operating system, we must first identify the points of contact between the default Windows 10 installation and files identified in the LOLBAS project described in Chapter 2.1.3.

The dual-use files included in default Windows 10 installation, unless specifically removed, are included in most operating systems and are signed by Microsoft. This section aims to find and analyze how many files from the LOLBAS project are included in the default Windows operating system installation. If and what kind of signatures the files have and analysis of the operating system based on the found information regarding signatures, to

better understand the situation for designing the policies.

4.4.1 Parsing filepaths from LOLBAS project

To find the subset of mentioned files, we can parse the LOLBAS project data [19]. The visual LOLBAS project web page is based on the files in the project Github page, which consists of YAML files [9]. With a bit of scripting in python, we can parse the YAML files and generate a CSV file that includes a list of LOLBAS entities and their possible paths in Windows OS. The 'LOLBAS-filepaths.py' script source code is added as Appendix 5.1.4.

At the time of writing, there are 158 entities listed in the LOLBAS project[19]. Not all files are included in the default Windows 10 Enterprise installation. However, many have at least two possible locations listed, either for 32-bit or 64-bit operating systems or for other reasons, for example, when the path includes a version number.

The script output consists of 488 file paths, of which seven were textual representations of the path not being fixed or available. The counts by LOLBAS entity types are shown in Table 1.

Table 1. *LOLBAS entity path count distribution by types.*

Type of entity	Count of Paths
OSBinaries	347
OSLibraries	27
OSScripts	18
OtherMSBinaries	89
Total	481

4.4.2 Finding dual-use files existing in default Windows installation

The list of paths from script 'LOLBAS-filepaths.py' from 4.4.1 chapter allows us to use PowerShell to check which of these files exist on the default Windows 10 operating system installation and to check if and by whom the files are signed. The 'Get-LOTLInfo.ps1' script source code is added as appendix 5.1.4.

The signature checking is done by utilizing PowerShell cmdlet Get-AuthenticodeSignature [24], which allows for gathering several important pieces of information about the files. First, we see the signature status and signers certificate information and secondly, we also get a Boolean property called 'IsOSBinary' representing with value 'True' that the file is a

part of operating system release [25][26].

When considering adopting the approach, the process is easily repeatable by using the python script and Powershell script provided in Chapter 5.1.4 to get more relevant results. First, create a CSV file with an up-to-date list of paths from the LOLBAS project [9] using python script, and then use output CSV to run Powershell script on your golden image used for deployment.

From the Table 2 it is visible that Windows 10 Enterprise default installation includes 185 LOLBAS files, all of which are signed by Microsoft. For all existing files, the Boolean value for 'IsOSBinary' was also 'True'.

Table 2. *LOLBAS files existence and signature*

Signer Subject	Signature verified	File not found	Total
CN=Microsoft Windows, O=Microsoft Corporation, L=Redmond, S=Washington, C=US	185	0	185
(blank)	0	303	303
Total	185	303	488

4.4.3 Other files in the system with same signer

When utilizing the signature to create block rules, it is essential to ensure that it would not be causing collateral damage - blocking other unwanted files with the same signature. It is possible to run a Powershell script to analyze the filesystem recursively and find all files with the same signature. One possible approach is to run following script with SYSTEM permissions:

```
1 Get-ChildItem c:\* -force -Directory -exclude 'Windows', '
   Program Files', 'Program Files (x86)' |
2 Where-Object -property Attributes -NotLike -value "*
   ReparsePoint*" |
3 Get-ChildItem -Recurse -File | Where-Object -property
   Attributes -NotLike -value "*ReparsePoint*" |
4 Where-Object {
5     (Get-AuthenticodeSignature -FilePath $_.FullName).
       SignerCertificate.Subject -eq 'CN=Microsoft Windows,
       O=Microsoft Corporation, L=Redmond, S=Washington, C=
```



```
        US'  
6     } |  
7     Select-Object FullName |  
8     Export-Csv -Path C:\Users\labuser\MS-Signed-files.csv
```

The script tries to get signature information for all files under path 'C:\', where the signer matches with the signer we found to be used for files identified in the LOLBAS project. The script also filters out Reparse Points to reduce access denied errors.

When analysing the results from the script executed on default Windows installation it is evident that the only extra location we need to consider is 'C:\ProgramData\', in addition to common locations 'C:\Windows\', 'C:\Program Files\', and 'C:\Program Files (x86)\'.

4.5 Application Control for Windows

Application Control enables administrators to control which applications are blocked and allow everything else - a blocklisting approach - similar to anti-virus signatures, but the rules can be much broader. Alternatively, administrators can create a list of allowed applications and block everything else instead - an allowlisting approach. Allowlisting takes out the guessing by endpoint protection algorithms if something is malicious or not - with Application Control, everything not necessary can just be blocked.

According to Microsoft, the Application Control allowlisting takes the approach to workstation security to different paradigm[27]. Application Control moves away from an application trust model where all applications are assumed trustworthy to one where applications must earn trust in order to run[27].

Theoretically, this paradigm change is actual. However, when implementing Application Control solutions, it is very likely that the default rules from Windows Defender Application Control base policies [2] and/or from AppLocker default rules [28] will be kept in place - meaning that anything signed by Microsoft or located in Windows or Program Files folders would be allowed. Making the Application Control useless against the attacks considered in this work unless specific rules are created to prevent this.

Application Control is a powerful tool for securing an IT environment, but it has some caveats too.

The fear of additional complications and workload to manage such lists and possible additional load to the helpdesk to solve any edge-cases that might pop up during the

deployment phase is likely the primary reason why Application Control is often neglected, especially by smaller organizations. IT is in constant change and will likely stay like that in the near future. There are ways to create rules based on paths and publishers or signers to make managing the rules simpler than managing hash-based rules, but the need to manage the rules remains.

It is true that implementing Application Control thoroughly and successfully needs more work than installing anti-virus software on the workstations. However, when approached methodically and carefully, it can bring significant benefits [29].

Microsoft has listed a few important points to note for successful deployments of Windows Defender Application Control to give ideas on what to consider beforehand [29]:

- Executive sponsorship and organizational buy-in is in place.
- There is a clear business objective for using application control, and it is not being planned as a purely technical problem from IT.
- The organization has a plan to handle potential helpdesk support requests for users who are blocked from running some apps.
- The organization has considered where application control can be most useful (for example, securing sensitive workloads or business functions) and also where it may be difficult to achieve (for example, developer workstations).

4.5.1 Cost of Application Control

The Application Control features are included in the Windows operating systems with enterprise-level editions or enterprise management solutions[30][31], without any additional licensing costs. The costs are transferred to the time system administrators spend designing, deploying, and managing the configurations.

According to an analysis from 2014, the upkeeping costs of Application Control are not higher than the costs of not using Application Control [18].

The exact numbers in the paper are outdated, but the general proportions of costs are comparable today. The calculations in Figure 1 are based on an organization with 800 workstations.

The initial set-up cost is more difficult to predict. It depends very largely on the existing skills of the system administrators, existing IT processes, the number of third-party applications used, and the complexity of IT environments, among other dependencies.

Cost type	No whitelisting	Whitelisting
No. of reimages vs. no. of rules	1 to 2 infections per week requiring reimaging of the computer	2 to 3 new applications per week requiring research and new rules
Cost per incident	US\$50 to reimage a computer	System administrator's rate of \$25 for 30 minutes
Lost productivity	\$50 of user time waiting for delivery of a loaner computer	\$25 of user time waiting for the engineer to whitelist an application
Annual cost	\$5,200 to \$10,400 plus greater risk of a major breach	\$5,200 to \$6,800

Figure 1. *Application Control cost comparison [18]*

The implementation and up-keeping costs of Application Control can be significantly reduced by the techniques provided by R. Durve and A. Bouridane in their paper from 2017[17].

The policies for Device Guard allowlisting can be automatically generated on a 'golden image' machine, a system pre-audited to be running only trusted software[17]. Powershell Cmdlets[32] can be utilized to automatically create rules based on software vendor signing information, where applicable, and fallback to hashes, to significantly reduce the need for manual updating Application Control rules with software updates[17].

It is important to note that although the paper by R. Durve and A. Bouridane[17] focuses on Device Guard, which is now defunct - the approach is still applicable to Windows Defender Application Control and AppLocker.

4.5.2 Windows Defender Application Control

Windows Defender Application Control (WDAC) is the newest feature of Windows regarding Application Control functionalities. It was introduced with Windows 10[33]. Earlier similar functionality was known as configurable code integrity or CCI[33]. WDAC was also one of the features of the now-defunct Device Guard [33].

The WDAC policies apply to the computer as a whole and do not take users into account[33]. However, WDAC is superior in including more capabilities - for example, per-app rules, Reputation-Based intelligence, multiple policy support, and driver files support[31]. The WDAC policies can be applied to workstations in several ways - via Mobile Device Management solutions, like Intune; Configuration Manager; Powershell, or Group Policy[33]. Important to note that the deployment via Group Policy is only possible for devices running the Enterprise edition of the operating system[33].

Microsoft is recommending using WDAC over its predecessor AppLocker whenever possible[33]. Although AppLocker is still receiving security fixes, it will not receive any feature updates[33]. The AppLocker and WDAC do have differences in the configuration options, and the current best practice from Microsoft is to deploy AppLocker as a complement to WDAC if necessary - the priority should be enforcing WDAC at the most restrictive level possible and then add AppLocker for further fine-tuning[33].

WDAC policy rules

Before continuing with the WDAC policy design, it is crucial to understand User Mode Code Integrity. Windows operating system has two components that enforce the trustworthiness of the system - Kernel Mode Code Integrity (KMCI) and User Mode Code Integrity (UMCI)[34][35].

KMCI ensures that code running in Windows kernel has been signed and trusted and ensures that Windows kernel has not been compromised before to operating system loading[35].

UMCI ensures that all subsequent code (application software executed after loading Windows operating system) is trusted[35].

WDAC is utilizing UMCI to enforce file rules, and UMCI must be enabled by enabling Rule Option 0. It is key to note that if Rule Option 0 is used to enable UMCI, but the policy does not include any rules for user-mode executables, then WDAC will block all applications, even critical Windows user session code[34].

With Audit mode enabled, such mistakes in policies are only visible in the growing amount of event logs, and it is always recommended to start testing with Audit mode.

WDAC file rule levels

The WDAC file rule allows administrators to specify the level at which to trust the applications in quite a lot of detail, too many to list all in the paper - but the most important for current use case are listed below[34]:

- Hash - most granular option and too resource-intensive from a management perspective, but will be used in Windows recommended block list.
- FileName - easier to manage than a list of hashes but offers less security. Other measures must be applied to restrict renaming.
- FilePath - similar to FileName, with little more granularity. It needs other measures

to restrict changing the path.

- Publisher - 'Publisher' at PcaCertificate level (typically one certificate below the root) and Common Name of the leaf certificate.
- FilePublisher - combines the 'FileName' attribute of the signed file, plus 'Publisher' at PcaCertificate level (typically one certificate below the root) and Common Name of the leaf certificate, plus a minimum version number.

Applicability of the WDAC rules

Although the file rule levels are granular and offer more variety than pointed out in the previous section, because the WDAC policy is applied at the machine level and cannot include exceptions by users, it limits the usability. Only the files for which there is very high confidence that any user, including system administrators, would not require it can comfortably be added to the policies. Other restrictions must be implemented using other methods.

A good starting point for creating such a WDAC rule set is from Microsoft and is called 'Windows recommended block rules'[36].

Precedence of the WDAC rules

The order the rules are applied in is based on built-in file rule conflict logic in WDAC[34]. The logic will first process all explicit deny rules, then process all explicit allow rules, and if none matched, then falls back to check extended attributes and Intelligent Security Graph[34].

WDAC rules include the implicit deny rule, which means that by deploying the example policies shown in the paper as enforced policies, it is very likely that some necessary third-party applications would be impacted.

For production scenarios, it is important to either start with stricter policy, analysis of logs, and running WDAC in Audit mode to build the policies with necessary allow rules, or to include FileName rules with "*" wildcard for both applications and drivers, as can be seen in *AllowAll.xml* example policy by Microsoft[2], and adding specific deny rules. Deny rules are still applicable, as deny rules have higher precedence.

Windows recommended block rules

The 'Windows recommended block rules' list is not something to take lightly. The list is created in collaboration by members of the security community, and neglecting this list in hand with less than ideal WDAC configuration might lead to ways to potentially bypass

the WDAC itself [36].

The recommendations list consists of already known binaries from the scope of this study from the LOLBAS project - but the recommendations list is not covering nearly all files from the LOLBAS project. Nevertheless, the list includes many other binaries in addition to the LOLBAS project. Making it an essential component of WDAC policy design.

Microsoft also provides an XML file in the article, which can be merged to existing policy [36]. As the 'Windows recommended block rules' include rules based on specific Windows versions, it is best to review the recommendations list before deploying new Windows versions to the production environment.

Base policies

If there is no existing WDAC policy to migrate the 'Windows recommended block rules' policy to, then it might be good to start from Microsoft-provided base policies. The descriptions of the various base policies available are listed on Microsoft documentation pages [2].

The base policy files can be found on Windows 10 Enterprise edition workstations, at 'C:\Windows\schemas\CodeIntegrity\ExamplePolicies\'.

To reduce the effort of creating exhaustive allowlisting base policy it is recommended to use approaches described in the section 4.5.1.

4.5.3 AppLocker

AppLocker was introduced with Windows 7, but it is still perfectly usable. Microsoft has stated that AppLocker will not be receiving feature updates, and the focus is now on Windows Defender Application Control[33]. However, security updates will still be provided for AppLocker, and even Microsoft best-practice suggests that there are valid use-cases for AppLocker today[33].

In addition to being in the phase-out state, the AppLocker depends on a service called 'Application Identity'. Although stopping the service should require administrative privileges, if the attacker finds a way to rename or corrupt the service files or disable the service's startup, the AppLocker would be disabled on the next operating system startup. Provides an additional reason to utilize Windows Defender Application Control where applicable.

The primary differentiator between WDAC and AppLocker is that WDAC is applied on the computer level. At the same time, AppLocker can be configured to apply to individual users or security groups and therefore providing more flexibility for system hardening in some situations.

Secondly, the AppLocker has an option to create a rule and then add exceptions to the rule - again, making it more flexible when compared to WDAC in some situations.

4.5.4 AppLocker rules

AppLocker rules can be created into five collections[37]:

- Executable files: .exe and .com
- Windows Installer files: .msi, .msp, and .mst
- Scripts: .ps1, .bat, .cmd, .vbs, and .js
- Packaged apps and packaged app installers: .appx
- DLLs: .dll and .ocx

And the rules can be based on three different conditions[37]:

- Publisher[38] - consists of Publisher, Product name, File name, and File version fields. Wildcard * can be used, but for the entire field only, not for partial matches. Only applicable for signed files.
- Path[39] - Full path is required. The wildcard * is allowed in the file path. Some path variables are available.
- File hash[40] - Any file can be assigned, but the rule must be updated each time the file changes.

The AppLocker design is based on implicit deny - in case you deploy an AppLocker policy with very specific deny or allow rules only, everything else would be restricted[41].

It is also important to note that deny rules have higher precedence than allow rules in AppLocker policies - this allows creating broader allow rules and overriding with specific deny rules within the allow rule[41].

AppLocker Publisher rule for blocking LOLBAS files in non-native locations

As the analysis showed in Section 4.4.2 and Section 4.4.3, the files signed similarly to the found files from the LOLBAS project, are located in a few standard folders, which need

our focus.

To create the restrictive policy rule, we can instead focus on all other locations. Most of the locations listed are not user-writable, meaning that standard users should not be able to modify these files, allowing us to utilize WDAC to block the LOLBAS binaries in their original locations. But would leave a security gap where users could copy the file from the original location to an alternative location and bypass the WDAC FilePath rules or copy and rename the file to bypass FileName rules.

Based on Publisher criteria, we can utilize AppLocker to create rules into Executable file, Scripts, and DLLs collections to mitigate the threat above, as the LOLBAS files in the system are all signed. The rule should be applied to a standard users security group and exclude administrative users. Administrative users likely need to use additional Microsoft signed tools. Administrative users could also be restricted to running such files from a specific folder for higher security.

The created Publisher rules would not apply any additional restrictions but rather enforce the restrictions from WDAC policy and not allow execution of Microsoft signed binaries and scripts from the locations where they should not be. Executing existing files on the system would not be blocked if the files were in the correct locations.

In the proof-of-concept approach, the Publisher rule is applied to files signed by Microsoft to apply a comparable approach that the Default rules provide, but for a few more file locations. Similarly, the same approach can be extended to binaries in the system signed by other trusted vendors.

At the time of writing, there is only one file listed in the LOLBAS project, which would not fall under any possible rule of AppLocker, due to the not supported file type - Manage-bde.wsf[42]. To add additional security layer Access Control Lists are discussed in Section 4.6

4.6 Access Control List

The third helpful feature to protect the Windows operating system from attacks from trusted sources is Access Control List (ACL). An ACL is a list consisting of zero or more Access Control Entries (ACE), and the ACEs control the security behavior associated with a given (protected) object[43].

In our scope of interest in ACL features, the objects of focus are files and folders in the

file system. Not all file systems support a granular approach as a per-file or per-directory security descriptor model, but all newer Windows 10 operating systems are using the NTFS file system, which does include the support[44]. In addition to files and folders, the target object can also be printers, registry keys, and Active Directory Domain Services (AD DS) objects[45].

The security descriptor consists of four distinct pieces[44]:

- **Owner** - The security identifier (SID) of the owner of the object - a fallback setting, the owner always has the ability to reset the security on the object. Even if the owner removes permissions to do any actions on the object, the owner still has inherent permissions to restore the security rights.
- **Group** - Optional SID of the default group. Not required in Windows, but useful for managing permissions in some situations.
- **SACL** - System Access Control List - describes the auditing policy of the security descriptor.
- **DAACL** - Discretionary Access Control List - describes the access policy of the security descriptor.

The Access Control List feature is significant because, as described earlier in the paper regarding Windows Defender Application Control in the section 4.5.2 and AppLocker in the section 4.5.3, the first applies on machine level and the second has limited file type support. There are situations where more control is required.

For example manage-bde.wsf - the trouble with manage-bde.wsf is that the tool itself is legitimate and useful tool for administrators, and it would not be best to block or remove the utility for everyone, which is why the Windows Defender Application Control and AppLocker are not suitable in this case. Full description in subsection 4.6.1.

4.6.1 Example of manage-bde.wsf

The manage-bde.wsf is a Windows component for managing the BitLocker disk-encryption feature - it can be used in place of the BitLocker Drive Encryption Control Panel item to turn BitLocker on or off, specify unlock mechanisms, update recovery methods and unlock BitLocker-protected data drives[46].

Manage-bde.wsf is also a part of the LOLBAS project[42] - as the same file can be used for MITRE ATT&CK technique T1216 Signed Script Proxy Execution - to potentially bypass application control and signature validation on systems[47]. The entries were added to

both LOLBAS and MITRE ATT&CK projects in 2018, but the approach is still working on the testing environment with the current Windows 10 Enterprise edition (Lab description in section 4.3). Example shown on Figure 2

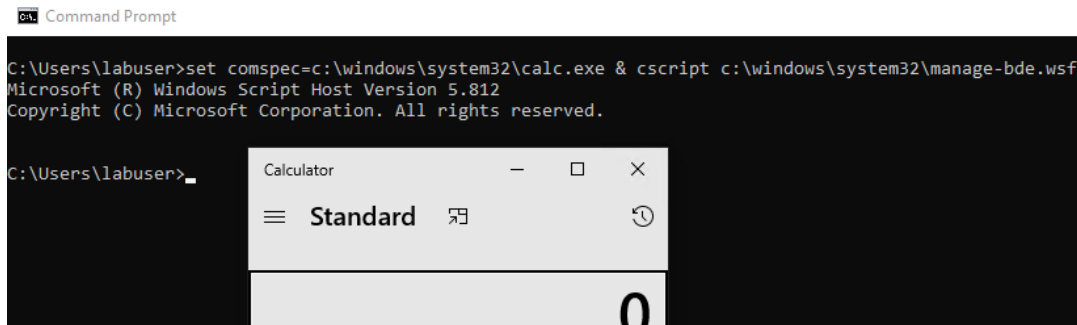


Figure 2. *Calc.exe* execution from *manage-bde.wsf*

The trouble with *manage-bde.wsf* is that the script itself is legitimate and useful tool, and it might not be best approach to block or remove the utility for everyone, which is why the Windows Defender Application Control is not suitable to solve the task at hand of hardening Windows operating system. AppLocker, on the other hand, is limited by the supported file types and not applicable.

4.6.2 Design of DACL rules

To implement an Access Control List - or specifically, to implement a Discretionary ACL or DACL to describe the access policy, we need first to understand the basics of DACLs and Access Control Entries (ACEs).

If a Windows object does not have a DACL, the system allows everyone full access to it. If an object has a DACL, the system allows only the access that is explicitly allowed by the ACEs in the DACL for this object[48].

The rules include implicit deny, meaning, if there are no ACEs in the DACL, the system does not allow access to anyone - or if there is a specific allow rule to a limited set of users or groups, the access is implicitly denied for everyone else[48]. Both Allow rules and Deny rules are available for fine-grained configuration [48].

To illustrate the working of DACL, Microsoft has created a good visualization, see Figure 3

Order of ACEs in a DACL

It is also important to note the order in which the different ACE rules are applied [49]:

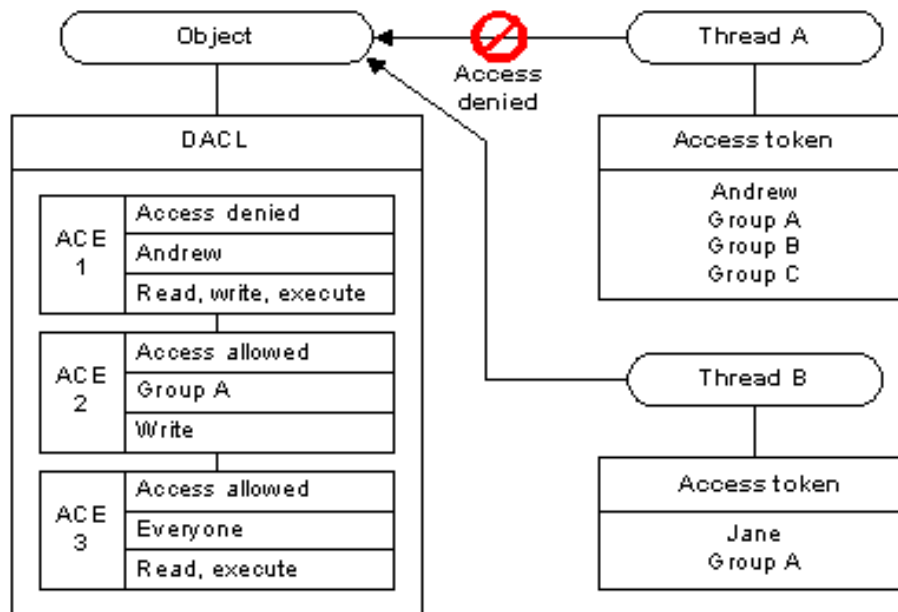


Figure 3. DACLs and ACEs [48]

- All explicit ACEs are placed in a group before any inherited ACEs.
- Within the group of explicit ACEs, access-denied ACEs are placed before access-allowed ACEs.
- Inherited ACEs are placed in the order they are inherited. ACEs inherited from the child object's parent come first, then ACEs inherited from the grandparent, and so on up the tree of objects.
- For each level of inherited ACEs, access-denied ACEs are placed before access-allowed ACEs.

The most critical piece of information to note is that deny rules have a higher order of precedence than allow rules. In case you miss this, and for clarity, try and add a Deny rule for "Everyone", in addition to specific Allow rules. Then you have rendered all other rules extraneous since the Deny rule is explicit and has the highest possible priority, overriding all other rules.

To have implicit deny for all other users applied, just an empty DACL set or single Allow rule will be sufficient.

4.7 Protecting against attacks via Powershell

Although the majority of the study focuses on the potentially dangerous binaries, scripts, and libraries pointed out by the LOLBAS project[19] and by Microsoft[36], there is an essential additional aspect to consider - Powershell.

There are two significant reasons Powershell needs to be considered in the context of the study at hand.

First, Powershell is a powerful tool for system administrators and malicious actors.

The use of Powershell has been increasing and is being used to implement attacks on various stages of real-world attacks while bringing challenges like heavy obfuscation of the code[50][51].

Powershell is listed as a distinct MITRE ATT&CK framework sub-technique, T1059.001 Command and Scripting Interpreter: PowerShell[52]. Powershell is, in some ways perfect tool for Living off the Land attacks - it exists by default on most current Windows systems, blends in with administration tools, and often leaves few artifacts[11]. Although, the logging is disabled in Windows by default, with some configuration, logging of Powershell can be comprehensive[51].

Powershell-written post-exploitation frameworks are, among other malicious uses, also utilized in cyber espionage operations, for example, PowerSploit Framework, PowerShell Empire Framework, and PoshC2 Framework[53].

Second, when implementing AppLocker or Windows Defender Application Control, system administrators are enabling Powershell Constrained Language mode by default unless manually disabling the feature.

For WDAC the protection is set by the default value of False for "*option 11*" - standing for "*Disabled:Script Enforcement*"[34], and automatically enforced when WDAC policy is set to Enforced mode.

4.7.1 Powershell Constrained Language mode

Powershell Constrained Language mode is one of four modes available for Powershell [54]. The modes determine what elements of Powershell are allowed in a given session[54].

The Constrained Language mode is designed to prevent using Powershell to circumvent the UMCI, which is the basis of Windows Defender Application Control[35][54].

The Constrained Mode allows to run signed scripts, a limited set of Cmdlets and perform simple administrative tasks and blocks COM objects, unapproved .NET types, dot sourcing, among other restrictions[55].

4.8 Implementing the rules

4.8.1 Layers of protection on high-level

The overall approach will be implemented in multiple layers to provide comprehensive protection against Living off the Land attacks.

Signed binaries from non-native locations

The first and broadest layer is blocking all binaries using the same signature we have found in the earlier analysis of LOLBAS binaries in Section 4.4.2. As this is a broad limitation, this would be only applied to standard users group and not to administrators - therefore, the technical measure for this approach is AppLocker.

This approach helps mitigate threats from known and trusted binaries introduced to the system by users or third parties. Attacks using trusted binaries already existing in the system are still possible and need to be mitigated using the following measures.

Blocking dangerous binaries for all users

The next layer of defense is to analyze the usage patterns of LOLBAS files in the production environment and utilize Windows Defender Application Control to block unnecessary binaries for all users, standard users, and administrators alike.

The restrictions introduced in this layer would be done using the file paths received from earlier parsing of LOLBAS project files in Section 4.4.1. Additionally, the Microsoft Recommended Block Rules are used to add additional restrictions [36] for potentially dangerous binaries and scripts.

Blocking potentially dangerous binaries for standard users

In this layer, we introduce additional restrictions, but only for the standard users' group, as the utilities are required for administrative tasks, and blocking the utilities can negatively affect performing the legitimate management tasks.

AppLocker can be used to create a user or group-based restrictions in the system.

Blocking edge-cases with Access Control List

The analysis in the design phase of the work presented an edge-case, where both Windows Defender Application Control and AppLocker have limitations and are not applicable.

Access Control List will be utilized to add more fine-grained restrictions based on users and groups.

Limiting untrusted binary execution for standard users

The previous methods provide protections against abuse of known and signed files in the system - there still exists a possibility that attackers would find a way to introduce the otherwise known and trusted binaries, but with removed signature and to a non-restricted path or with a modified filename.

To mitigate such threats, generic Application Control allowlisting should be applied for standard users, limiting execution to files with trusted signatures and from locations that are not user-writable. The allowlisting is the default approach of AppLocker and WDAC. Disabling this requires additional rules added manually.

Neglecting the allowlisting method can make bypassing the restrictions above trivial and should not be taken lightly. The protection would likely still apply for the majority of automated attacks but not for advanced threats. As described in Section 4.2.1, the suitable target organizations for the approach have to be with mature IT processes and security, where implementing allowlisting should not be remarkably challenging.

4.8.2 Implementing Windows Defender Application Control

To deploy the Windows Defender Application Control policy to endpoints, the administrator needs to go through the following steps, as an ongoing process to improve the configuration through iterations:

1. Create or collect XML files describing the WDAC policy
2. Merge the XML files, if WDAC policy consists of multiple files
3. Convert the XML file to binary format
4. Make binary file accessible for endpoints
5. Configure endpoints to apply WDAC policy from binary file
6. Analyse WDAC logs

The best practice would be to store the input files in version control for safe storage, versioning, and change tracking. Some of the steps above can be automated, and a proof-of-concept approach is provided below.

The approach described below is not intended to be plug-and-play automation but a simple

proof-of-concept to show one possible basic approach as a starting point to develop an automated deployment solution.

Create and collect XML files

To create the proof-of-concept WDAC policy for this paper, Microsoft Example Base Policy *DefaultWindows.xml* in Audit mode will be used as a starting point[2].

In a production deployment, a policy describing other business applications in use should be merged too. See section 4.5.1 for the approach to simplify the creation of such policy. This policy can substitute or supplement the Microsoft Example Base Policy.

The base policy is merged with Microsoft recommended block rules [36].

Finally, a custom script *LOLBAS_WDAC-policy.py* (Appendix 5.1.4) is utilized to generate a third XML file, which contains the paths of LOLBAS binaries to be blocked by WDAC.

The input CSV file for the *LOLBAS_WDAC-policy.py* script is the output from script *LOLBAS-filepaths.py* shown in Appendix 5.1.4 - the output CSV needs to be created with switch '-p' to add additional column called 'Block in WDAC/AppLocker', where administrator can mark items to block with keyword 'WDAC' to be included in WDAC policy.

The script *LOLBAS-filepaths.py* is described in section 4.4.1.

Only rows with 'WDAC' marking will be included in the WDAC Policy XML created by *LOLBAS_WDAC-policy.py*.

Merge the XML files

Microsoft provides a Powershell Cmdlet `Merge-CIPolicy` [56], described in Microsoft documentation for WDAC[57].

The Powershell Cmdlet makes the process easily automatable. It provides an easy way to split the WDAC policies into different files based on the purpose, simplifying the management and versioning.

To merge multiple XML files, for example following command can be used:

```
Merge-CIPolicy -PolicyPaths .\DefaultWindows_Audit.xml,.\
  WDAC_policy_20220417_102736.xml,.\MS-recommended-block-rules.
```

```
xml -OutputFilePath WDAC_Policy_v0.1.xml
```

Audit or Enforce mode

The default option for WDAC is having *Audit Mode* enabled - the behavior is controlled by Rule option 3 'Enabled: Audit Mode'[23]. To enforce the WDAC policy, delete the option from the policy XML file[23].

Convert the XML file to binary format

After combining the required XML files into a single entity, it is possible to convert the XML file into binary format to prepare it for application to endpoints.

The conversion can be automated by using Powershell Cmdlet *ConvertFrom-CIPolicy* [58].

For example, the following command can be used: To merge multiple XML files, for example following command can be used:

```
ConvertFrom-CIPolicy -XmlFilePath .\WDAC_Policy_v0.1.xml -  
BinaryFilePath WDAC_binary.bin
```

Make binary file accessible for endpoints

Based on the infrastructure and network limitations, the administrator can either make the binary policy file available via Universal Naming Convention (UNC) path or deploy the binary file directly onto the endpoint.

Create Group Policy to apply WDAC

To apply WDAC policy to the endpoints administrator must create a new Group Policy Object (GPO) or modify existing one to include following setting[59] as shown on Figure 4:

```
Computer Configuration > Policies > Administrative Templates >  
System > Device Guard > Deploy Code Integrity Policy
```

The first sign of successful deployment of WDAC to endpoints is Windows log event with ID 7010 in *Microsoft-Windows-DeviceGuard/Operational*, as shown in Figure 5:

Analyse WDAC logs

After deploying the WDAC to the endpoints, it is vital to analyze the results of the applied WDAC policy to notice any usage of the blocked binaries and decide if the usage was

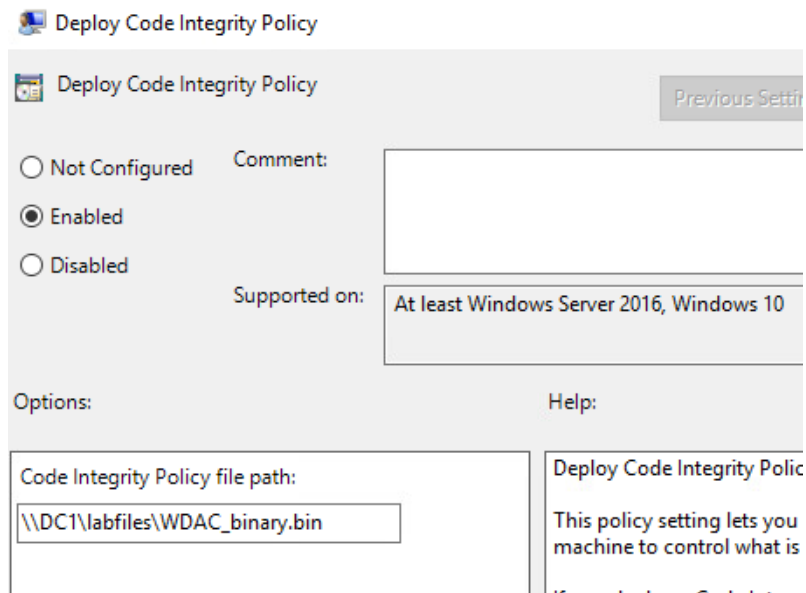


Figure 4. *Deploy Code Integrity Policy*

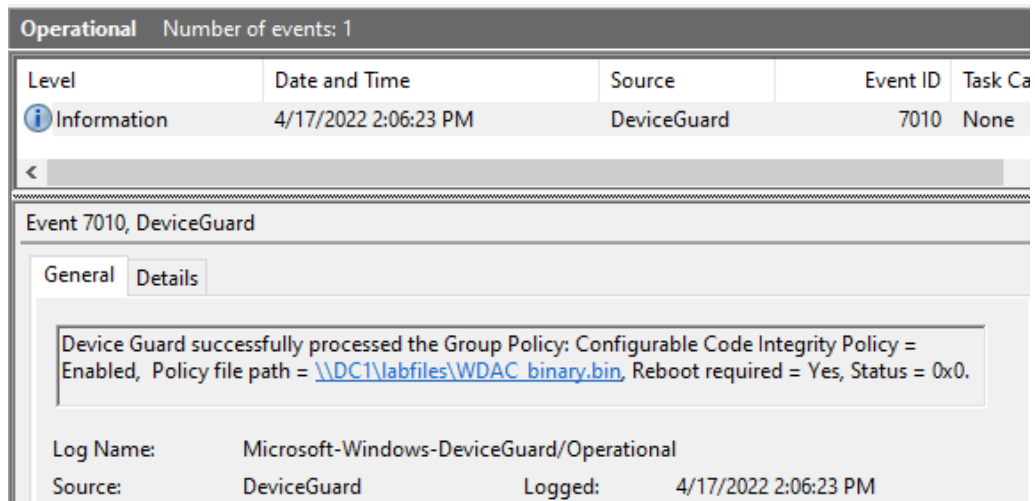


Figure 5. *Event Viewer - 7010 - Enabled Code Integrity Policy*

legitimate.

To analyse the logs, system administrators should focus on following events in *Microsoft-Windows-DeviceGuard/Operational* log[60]:

- 3076 - This event is the main WDAC block event for audit mode policies. It indicates that the file would have been blocked if the WDAC policy had been enforced.
- 3077 - This event is the main WDAC block event for enforced policies. It indicates that the file did not pass WDAC policy and was blocked.

Also, WDAC can generate events to the *Microsoft-Windows-DeviceGuard/Operational*

log[60]:

- 8028 - This event indicates that a script host, such as PowerShell, queried WDAC about a file the script host was about to run. Since the WDAC policy was in audit mode, the script or MSI file should have run.
- 8029 - This event is the enforcement mode equivalent of event 8028 described above.

Legitimately used binaries should be excluded from the next WDAC policy iteration. Binaries blocked with WDAC, which administrators require, could be moved from WDAC policy to AppLocker policy, which allows the user or group-based restrictions. As the logs present the name of the user using the binaries, it should be easily verifiable if the employee should be doing administrative tasks.

Malicious use of the binaries, where WDAC is working as intended, are indications of possible code execution to get an initial foothold by threat actors or indicate an attack in its later stages. Such events should be forwarded to the organization's Cyber Security Incident Response Team (CSIRT), if applicable.

After analyzing the logs, go back to step 2, described in 4.8.2, to modify the input files used for policy generation automation processes and repeat the WDAC implementation process.

4.8.3 Implementing AppLocker

The process of implementing AppLocker can be done using the following steps and should be implemented and improved in iterations:

- Create the XML file describing the AppLocker policy
- Deploy AppLocker policy
- Analyse AppLocker logs

Blocking signed binaries from non-native locations

To enforce the limitations described in subsection 4.8.1, limiting the use of signed binaries from non-native locations, we can utilize the Publisher rule described in the Section 4.5.4.

With AppLocker Publisher rule type we can create rules for each of the five collections listed in Section 4.5.4 to block everything with Publisher Name "O=MICROSOFT CORPORATION, L=REDMOND, S=WASHINGTON, C=US", as the found signer of

LOLBAS files in Section 4.4.2 - but add the exception for known legitimate locations, 'C:\ProgramData\', 'C:\Windows\', 'C:\Program Files\', 'C:\Program Files (x86)\', as found in Section 4.4.3.

To see an example of the AppLocker Publisher policy, see Appendix 5.1.4.

The example AppLocker policy above will also be the basis for further rules for specific paths or file names added to implement the protection layer described in subsection 4.8.1, blocking potentially dangerous binaries for standard users.

Blocking potentially dangerous binaries for standard users

After adding the Publisher rule, the next objective is to add rules based on file names because file paths and names of potentially dangerous files are the data provided by the LOLBAS project. These rules are targeted to specific users or groups in the Windows environment, therefore also allowing us to set restrictions on the utilities and tools needed by administrative users but not required by standard users.

Adding additional, more specific rules to the AppLocker policy will divide the rules into five Collections based on the supported file types (read more in Subsection 4.5.4).

A python script is created to simplify the AppLocker policy creation process and allow for an automated management process. The '*LOLBAS_AppLocker-policy.py*' script parses the same CSV input file created by script '*LOLBAS-filepaths.py*' - similarly to WDAC Policy creation in 4.8.2.

The script searches for rows with marking 'AppLocker', parses the file path to find the correct Collection to put the rule into, and outputs an XML policy file from generated rules. The script can be found from Appendix 5.1.4. Example CSV file used as input to the script can be found from Github[61].

Deploying AppLocker policy

Microsoft has not publicly documented any methods for fine-grained control to insert AppLocker rules in an automated manner.

Only options available are *Import Policy* and *Export Policy* functions in the graphical interface for AppLocker configuration (see Figure 6) and Powershell Cmdlets *Get-AppLockerPolicy* and *Set-AppLockerPolicy*. Resulting in need for some custom scripting to add rules to block LOLBAS binaries in automated fashion.

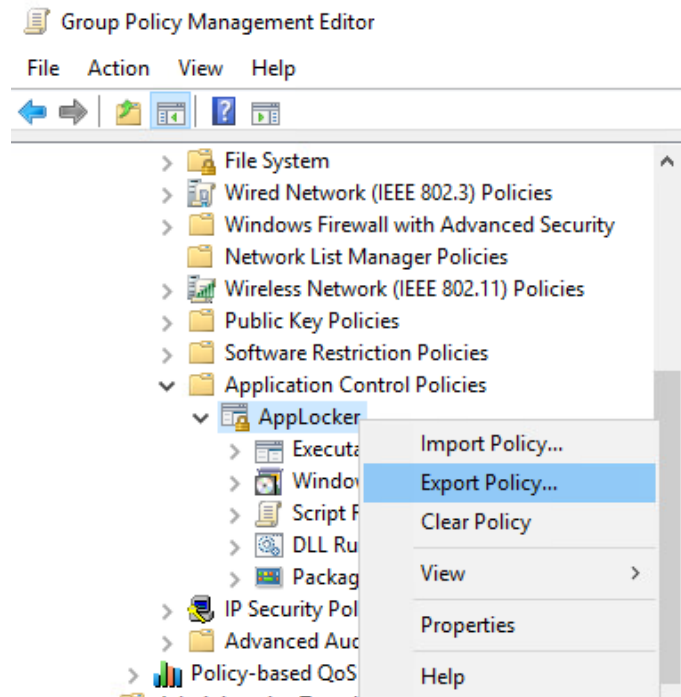


Figure 6. AppLocker UI - Export and Import

To deploy the generated AppLocker policy, it is important to have some base policy to migrate the new rules to, or at least include the AppLocker default rules - due to the AppLocker implicit deny design. Design of AppLocker is further described in 4.5.4.

To avoid accidental policy creation without Microsoft default rules for AppLocker the script *'LOLBAS_AppLocker-policy.py'* in Appendix 5.1.4 includes the default rules, unless manually overridden with switch *'-excludedefaults'*.

As mentioned in the subsection 4.8.3 the AppLocker policy with the Publisher rule is used as a template. To generate a policy file without the Publisher rule included use switch *'-excludepublisher'*.

After importing Policy with Microsoft default rules, Publisher rule with exceptions, and specific deny rules for LOLBAS files to the Group Policy Management Editor, the policy should look like in Figure 7

The current implementation of script *'LOLBAS_AppLocker-policy.py'* does not enable the AppLocker policy by default.

The policy must be manually enabled by going to *'AppLocker Properties'*, ticking *'Configured'* per Rule Collection, and selecting *'Enforce rules'* or *'Audit only'* - the latter option suggested as the first step. See Figure 8.

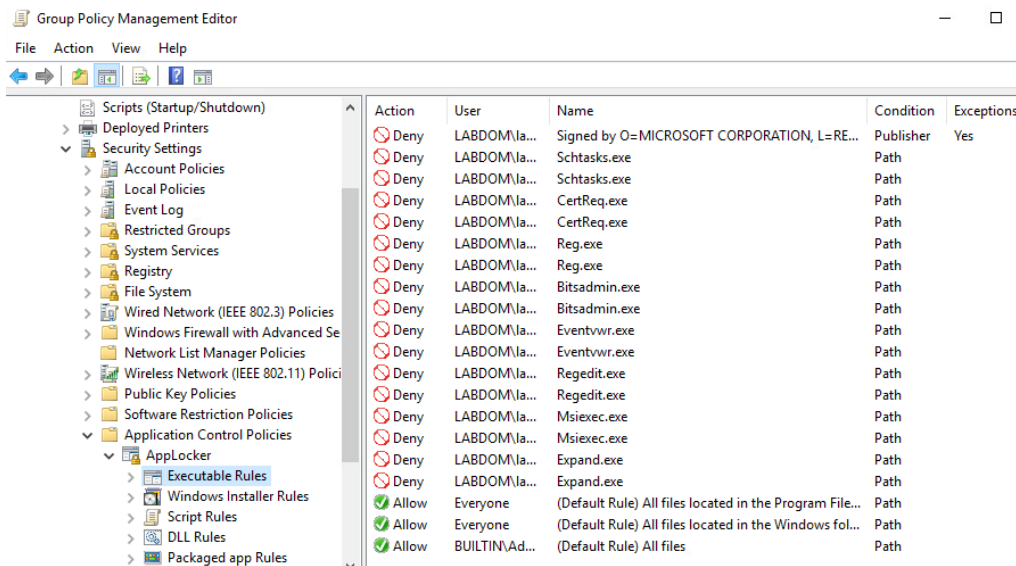


Figure 7. Group Policy Management Editor - AppLocker Implemented

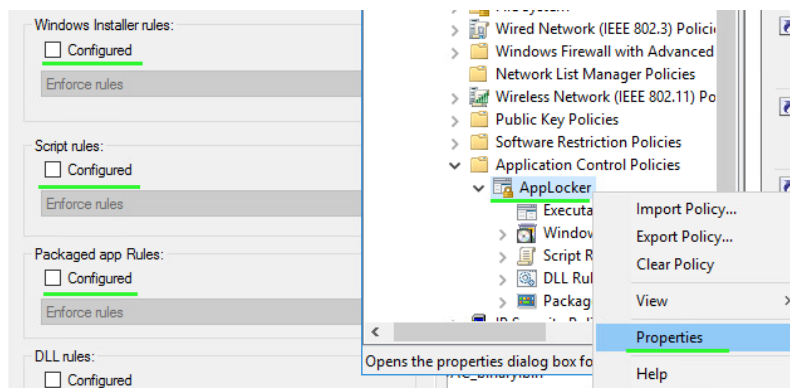


Figure 8. AppLocker Properties - Enable AppLocker

AppLocker is Enforced by default

Similarly, it is critical to take note of the warning message when configuring AppLocker Group Policy - "If rule enforcement has not been configured, rules will be enforced by default."

Meaning that in case the administrator plans to import the policy to Group Policy, which is linked to some target - and be safe, because administrator did not tick the 'Configured' checkbox - then actually administrator might just have blocked some applications, libraries or scripts without the intention to do so.

Since this is described in the GUI, this certainly must be a feature rather than a bug. See Figure 9

Such an approach might be implemented by Microsoft to allow for multiple Group Policy



Figure 9. *AppLocker Properties - Enforce AppLocker*

Objects to be applied simultaneously for AppLocker, but only one of those GPO-s setting the configuration of AppLocker being in Audit or Enforce mode to avoid misconfigurations.

AppLocker dependency - Application Identity service

It is also critical to enable *Application Identity* service on the workstations where AppLocker policy is deployed; failing to do so will not allow AppLocker to function, as *Application Identity* service is used to verify attributes as a file[62].

Enabling *Application Identity* can be done by Group Policy by changing the following setting:

Computer Configuration > Windows Settings > Security Settings > System Services > Application Identity > Properties > Start Automatically.

Analyse AppLocker logs

After deploying AppLocker to the endpoints, the event logs should be collected and analyzed to notice any misconfigurations and improve the AppLocker policies by iterations.

The AppLocker logs can be found in *Microsoft-Windows-DeviceGuard/Operational*. The events of interest should be the following:

- 8003 - *<File name> * was allowed to run but would have been prevented from running if the AppLocker policy were enforced. Applicable with "Audit only" mode of AppLocker. Applies to .exe or .dll files.
- 8004 - *<File name> * was not allowed to run. Applicable with "Enforce rules" mode of AppLocker. Applies to .exe or .dll files.
- 8006 - *<File name> * was allowed to run but would have been prevented from running if the AppLocker policy were enforced. Applicable with "Audit only" mode of AppLocker. Applies to script or .msi files.

- 8007 - *<File name> * was not allowed to run. Applicable with "Enforce rules" mode of AppLocker. Applies to script or .msi files.

After analyzing the logs, improve the input to policy creation automation, and re-deploy the AppLocker policy.

4.8.4 Implementing Access Control List

Access Control List is a powerful tool to cover the edge-cases falling out of the scope of Windows Defender Application Control and AppLocker. At the time of writing, the LOLBAS project includes only a single file, which falls into the edge-cases category in the paper at hand. For more information review section 4.6.1.

Creating DACL rules

The procedure for centrally managing DACL rules is not too evident in the Microsoft documentation on the ACL and DACL itself. However, with a little exploration and testing of Group Policy Management Editor console it is clear, that the File and Folder DACL rules can be managed from *Computer Configuration - Policies - Windows Settings - Security Settings - File System* menu, where administrator can add file or folder specific rules, as shown on Figure 10.

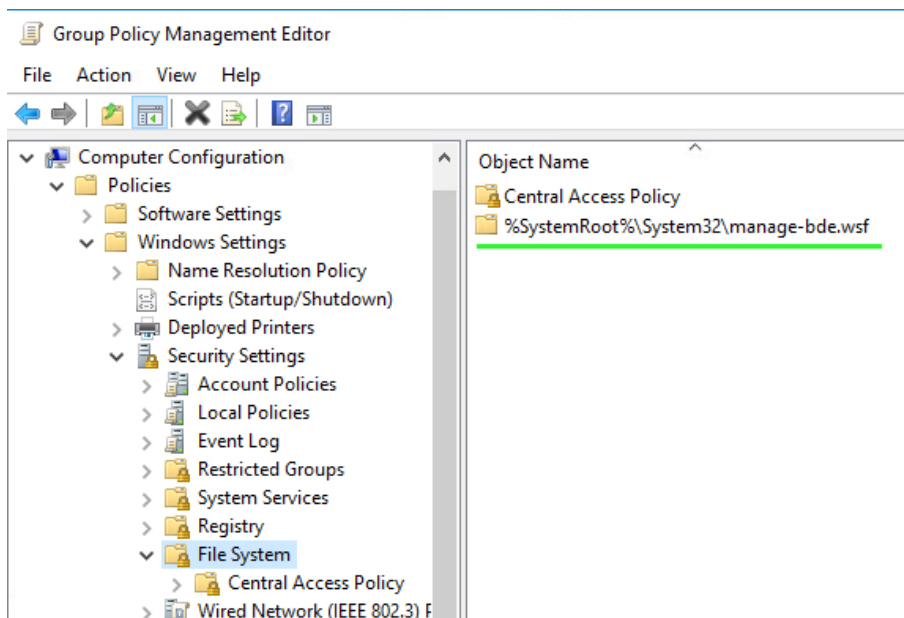


Figure 10. Group Policy Management Editor - File System

Blocking manage-bde.wsf

As described in subsection 4.6.1, the *manage-bde.wsf* is a legitimate and helpful script; therefore, the blocking rule might not be applicable to administrators of the workstation. Also, as discussed earlier, it is crucial to not add any broad scope Deny rules, as Allowing for specific groups introduces automatic implicit Deny rule for everyone else.

For example, to allow access only for users of the local Administrators group and deny it for everyone else, we need to create the rules shown in Figure 11.

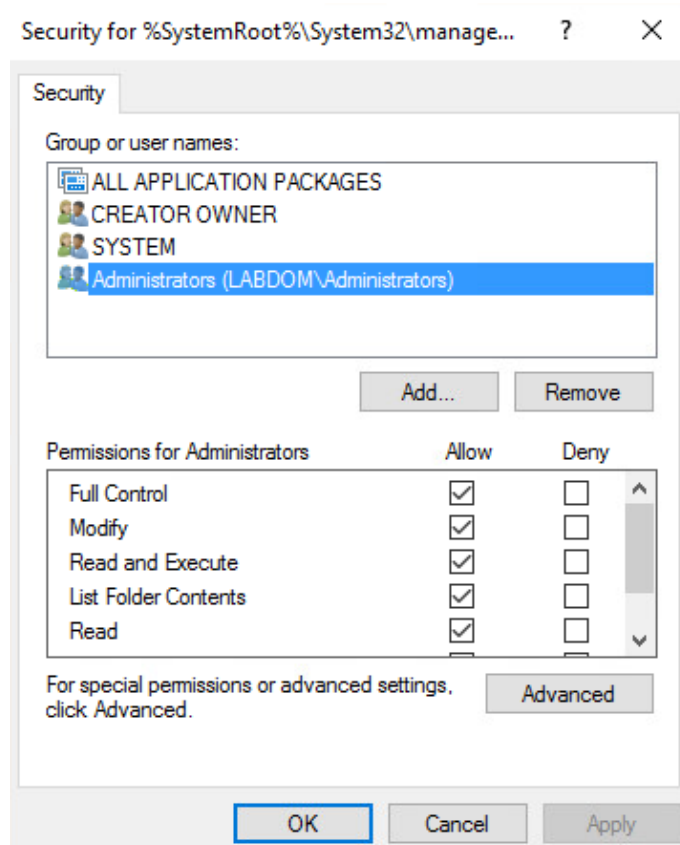


Figure 11. DACL - *manage-bde.wsf*

4.9 Testing the rules

The testing will be conducted by feature sets separately to isolate the features, and by conducting at least single test per rule type or rule Collection.

For testing, two identical fresh Windows 10 Enterprise installations were used, deployed as described in the lab description section 4.3.

The only difference between the machine for testing and the machine for control is the group policies - both have the default group policy as described in the section 4.3. The

testing machine has additional group policies applied, enabling the security features as described in the implementation section 4.8.

4.9.1 Testing WDAC policies

The WDAC policies are applied on the machine level and should not be affected by user accounts or the level of permissions. The rules are based on file paths, limiting only the use of files in their original locations.

WDAC, Mshta.exe

The Mshta.exe file is used by Windows to execute html applications (.hta)[63]. The Mshta.exe can be used to execute code either from .hta files, where code is embedded to files as JavaScript, JScript, or VBScript[63]. Moreover, the Mshta.exe can also be used to execute script code directly from command line arguments without using .hta files[63].

The use of Mshta.exe is listed as a distinct MITRE ATT&CK sub-technique under the Defense Evasion tactic and provides a large list of threat actors and tools utilizing Mshta.exe file for malicious purposes[64]. Mshta.exe can be used for bypassing Application control and Digital Certificate Validation[64].

Mshta.exe can also be found in the "Microsoft recommended block rules" in WDAC documentation[36].

The Mshta.exe has been known at least since 2015 when PowerShell-based remote access tool took advantage of .hta files[65] to establish a reverse shell.

Test case 1 - standard user, blocked file, native location

To execute tests with Mshta.exe we can execute following command to execute calc.exe process with potentially malicious pattern, by using the command:

```
mshta.exe javascript:a=GetObject("script:https://raw.githubusercontent.com/LOLBAS-Project/LOLBAS/master/OSBinaries/Payload/Mshta_calc.sct").Exec();close();
```

As a result, we can see cmd.exe output *"The system cannot execute the specified program."* and an event with ID 3077 in Windows logs, with the following message:

```
"Code Integrity determined that a process (\Device\HarddiskVolume4\Windows\System32\cmd.exe)
```

attempted to load \Device\HarddiskVolume4\Windows\System32\mshta.exe that did not meet the Enterprise signing level requirements or violated code integrity policy."

Illustrating successful deployment of WDAC rules for standard users.

Test case 2 - administrator, blocked file, native location

This test case is identical to Testcase 1, except for the user context executing the command.

As the WDAC rules are deployed on the machine level and do not depend on the user context, it is expected to see the same results as for Testcase 1, and the Figure 12 below presents this to be true.

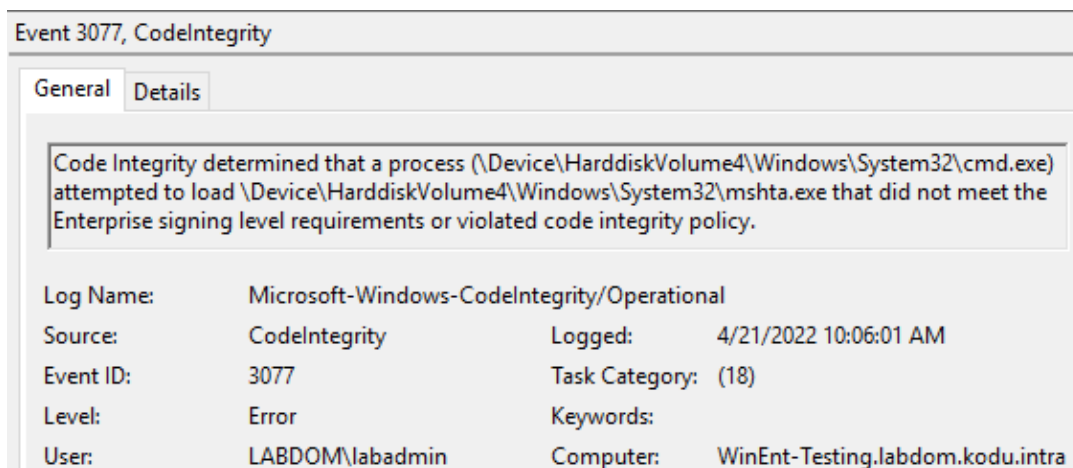


Figure 12. *Event Viewer - WDAC Event ID 3077*

4.9.2 Testing AppLocker policies

The AppLocker policies are used as an additional layer on top of WDAC rules. AppLocker provides more flexibility to configure per-user or per-group restrictions to allow blocking tools needed by administrators but poses a threat when used maliciously under the context of a standard user.

Additionally, AppLocker helps to prevent simple relocating or renaming of signed binaries, to avoid path-based rules.

Files used for testing

For testing the rules below, we will be utilizing three types of files to cover three rule collections of AppLocker. Executables collection, Scripts collection, and DLLs collection.

Executables collection

For Executables collection, we will use *schtasks.exe* - Windows Task Scheduler. *Schtasks.exe* is used to perform task scheduling. The utility can be run directly on the command line or by utilizing Graphical User Interface (GUI) from Control Panel[66]. The *schtasks.exe* can be used for malicious purposes to execute tasks on a startup or scheduled basis for persistence[67]. With administrative privileges, also for lateral movement[66].

The *schtasks.exe* binary is listed in the LOLBAS project, which has contributed to the generation of example policy for AppLocker [67].

Scripts collection

To test Scripts collection *pester.bat* will be used - the *pester.bat* is a part of Powershell *pester*, which is a testing and mocking framework built for Powershell[68] and as seen by the previous data gathering, included in the default installation of Windows.

The *pester.bat* script is listed in LOLBAS project[69] and is linked to MITRE ATT&CK technique T1216 Signed Script Proxy Execution and allows to bypass application control and digital certificate validation[70].

DLLs collection

Testing for DLLs collection will be done utilising native DLL file called *advpack.dll*. The *advpack.dll* is an utility for installing software and drivers with *rundll32.exe* [71].

The use of *rundll32.exe* file to execute arbitrary functions in DLL files is listed as specific MITRE ATT&CK sub technique T1218.011, under technique Signed Binary Proxy Execution - the technique potentially allows for anti-virus, application control, and digital certificate validation bypass[72].

True negative testing

For testing true negative situations we can use the trusted "*C:\Windows\System32\calc.exe*" binary for Calculator application, which has identical Authenticode signature[24] as other files used for testing.

Testcase 3 - AppLocker, trusted signed binary, native location

To start verifying the results of designed AppLocker policies, let us start with true negative testing - a situation where no policy is blocking the action, and no policy should be blocking the action either.

For this, we have selected known trusted Windows binary calc.exe, and we can execute the process by using the command line:

```
cmd /c "C:\Windows\System32\calc.exe"
```

As a result, we see the Calculator application opening, and from the Windows event log, we can find a matching AppLocker log with ID 8002, and with the message "*%SYSTEM32%\CALC.EXE was allowed to run.*".

As expected from designed policies - signed and trusted binaries, outside the scope of specific deny rules, and in their native locations have no restrictions applied.

Testcase 4 - AppLocker, trusted signed binary, relocated, blocked by policy

As described in section 4.8.3 and 4.8.1 we created a policy to block the use of signed Microsoft binaries in other than native locations.

It is vital to avoid easy security policy bypassing by simply copying the original file to an alternative location or introducing copies of blocked files by other means.

To test the policy we can copy the "*C:\Windows\System32\calc.exe*" binary to the test user desktop, and try to execute the same file again, by utilising following commands:

```
copy C:\Windows\System32\calc.exe C:\Users\labuser\Desktop\calc.exe
```

```
cmd /c "C:\Users\labuser\Desktop\calc.exe"
```

Based on the Publisher rule in the AppLocker policy, this time, event ID 8003 was logged to the Windows event log, with the message "*%OSDRIVE%\USERS\LABUSER\DESKTOP\CALC.EXE was allowed to run but would have been prevented from running if the AppLocker policy were enforced.*". See Figure 13.

The publisher rule is working as intended.

Testcase 5 - AppLocker, trusted signed binary, native location, blocked by policy

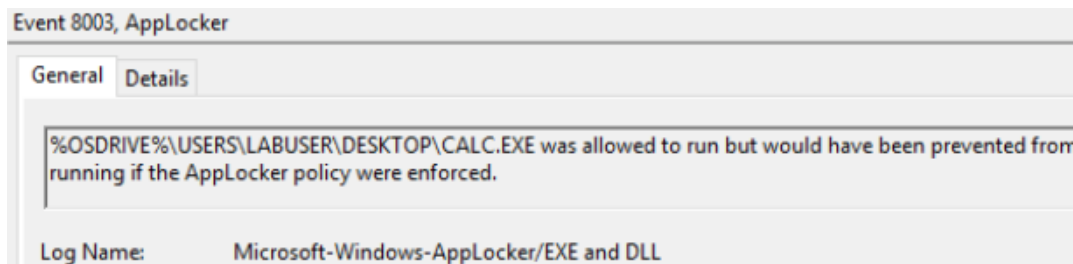


Figure 13. *Event Viewer - AppLocker ID 8003, calc.exe*

With previous tests covered, the mechanisms enforce the specific deny rules, now testing the deny rules for the potentially dangerous binaries can follow.

To execute the tests, one per rule collection, with files listed in 4.9.2 following commands can be used:

```
"schtasks /create /sc minute /mo 1 /tn 'Reverse shell' /tr c:\some\directory\revshell.exe"
```

```
"c:\Program Files\WindowsPowerShell\Modules\Pester\3.4.0\bin\Pester.bat" -? "$null; calc.exe"
```

```
"rundll32 advpack.dll, RegisterOCX "cmd.exe /c calc.exe"
```

The tests above give expected results based on the AppLocker policy design. Executable and DLL collection tests generate an event with ID 8003 to "Microsoft-Windows-AppLocker/Exe and DLL" log.

Scripts collection test generates an event with ID 8006 to "Microsoft-Windows-AppLocker/MSI and Script" log.

Testcase 6 - AppLocker, valid binary with invalidated signature, relocated

To test a possible situation where a malicious actor tries to break the signature of a known system file, to avoid the Publisher rule of AppLocker from applying, a system binary was modified.

The calc.exe, Windows Calculator executable, was modified by a single byte at the end of the file to change the file hash with minimal change in functionality, which would also make the signature of the file invalid. The file was saved as calc2.exe, transferred back to the testing system, and checked for signature info with Get-AuthenticodeSignature[24]. See Figure 14.

```
PS C:\Users\labuser> Get-AuthenticodeSignature .\Desktop\calc2.exe |
Select-Object Path, Status
Path                               Status
----                               -
C:\Users\labuser\Desktop\calc2.exe NotSigned
```

Figure 14. *Get-AuthenticodeSignature, calc2.exe*

After executing the modified Calculator app, *calc2.exe* - an event with ID 8003 was generated to AppLocker logs, showing that the *calc2.exe* file would have been blocked if AppLocker was in Enforce mode.

Although we do not have a specific rule to block unsigned binaries in the AppLocker policy, it is essential to note that there is implicit deny in the policy, when the policy is enabled. Everything not allowed would be blocked, as described in 4.5.3.

4.9.3 Testing the DACL for *manage-bde.wsf*

To test the effect of introduced DACL rule from section 4.8.4, to only allow access to *manage-bde.wsf* for administrative users, we can simply repeat the example command from subsection 4.6.1 explaining the problem, with different user contexts.

Test case 7 - Access Control List, standard user, native location

To test the Discretionary Access Control List rule for *manage-bde.wsf*, following command was executed in *labuser* context:

```
"set comspec=c:\windows\system32\calc.exe & cscript c:\windows\system32\manage-
bde.wsf"
```

As a result, error message was shown with message *"Input Error: Can not find script file "c:\windows\system32\manage-bde.wsf"*

Demonstrating that the blocking rule is working as intended.

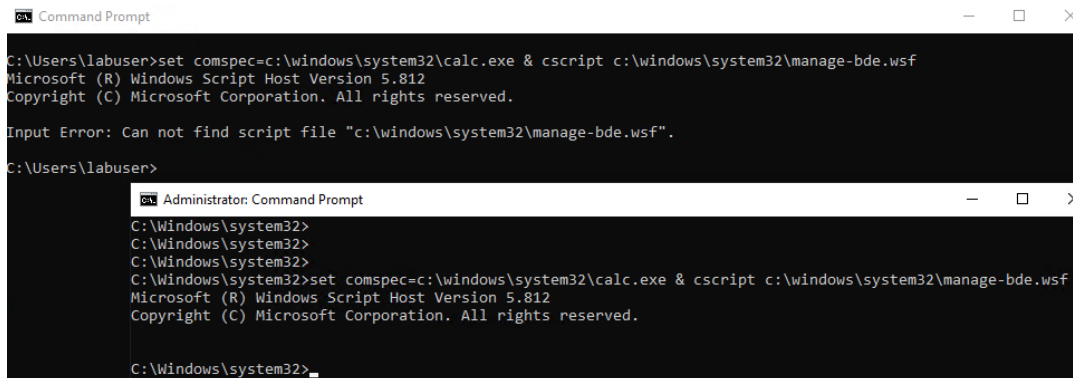
Since the rule is purely based on the path, then easy bypass is to copy the file to an alternative location. This is something that the restricted user can not do by copying the blocked file from the native location because the read access of the file is blocked - but the threat actor could ingest the file from alternative sources.

Test case 8 - Access Control List, administrator, native location

Testing the Access Control List rule with workstation administrator user, domain user *labadmin*, followed the same procedure used in Testcase 7 - with the only difference being the context of the user.

Script *manage-bde.wsf* was executed successfully - executing *calc.exe* process in our test, which was the expected result.

The figure 15 shows executing the same command with standard user permissions and with administrative permissions, with the expected results of allowing access to *manage-bde.wsf* file only with elevated privileges.



```
Command Prompt
C:\Users\labuser>set comspec=c:\windows\system32\calc.exe & cscript c:\windows\system32\manage-bde.wsf
Microsoft (R) Windows Script Host Version 5.812
Copyright (C) Microsoft Corporation. All rights reserved.
Input Error: Can not find script file "c:\windows\system32\manage-bde.wsf".
C:\Users\labuser>

Administrator: Command Prompt
C:\Windows\system32>
C:\Windows\system32>
C:\Windows\system32>
C:\Windows\system32>set comspec=c:\windows\system32\calc.exe & cscript c:\windows\system32\manage-bde.wsf
Microsoft (R) Windows Script Host Version 5.812
Copyright (C) Microsoft Corporation. All rights reserved.
C:\Windows\system32>
```

Figure 15. DACL tests - *manage-bde.wsf*

4.10 Results of the study

The single results from the eight test cases described in section 4.9 demonstrate that the implemented rules for Windows Defender Application Control, AppLocker, and Access Control List security features of Windows are working as intended.

Perhaps more importantly for the study at hand, the successful tests also show that the approach in general and the proof-of-concept scripts are performing in generating input files for Windows Security features. It displays the potential of creating semi-automatic processes to roll out measures to protect Windows systems against Living off the Land attacks.

Clearly, the provided scripts are not ideally suitable for enterprise-level deployments, which has not been the intention, as the goal has been to provide a methodology. The scripts can be used as a foundation to build the core of the tool to cover the methodology described in 4.10.3 and to include some of the additional features listed in 4.10.4.

4.10.1 Limitations of the defense

The Microsoft recommended files to block[36] together with the LOLBAS project[19], dedicated to finding and documenting dual-use binaries in the Windows operating system, provide a substantial list of files to block. Nevertheless, there is no guarantee that additional binaries will not exist to be abused and used to bypass the application control mechanisms.

Similarly, to reasonably allow the trusted binaries, the policies must use publisher rules, perhaps some path-based wildcard rules for exceptions too. These rules are prone to be abused too. If some exception is added to a user-writable location or some software vendors' signing certificates are stolen, it can provide a method for bypassing the defense measures described.

4.10.2 Meeting the set objectives

The first objective set for the study was to gather technical data regarding Living off the Land attacks and collecting data from Windows operating system. The step has been completed successfully and has provided valuable input for the second objective. This is covered by Section 2.1 and 4.4.

The second objective set was analyzing defensive measures in Windows operating system, understanding the features' capabilities, principles, and limitations, and designing a comprehensive set of security policies to defend against Living off the Land attacks. The subject is covered in Sections 4.5, 4.6, 4.7, and 4.8.

The third objective of providing a methodology on how to build a solution to defend against Living off the Land attacks is met and summarised as a list of steps to follow as foundations for building the semi-automated process. Each step concludes the earlier information and presents the procedures in a concise manner. The following section 4.10.3 will describe the procedures. Additionally, the scripts created and used as proof of concept are added as five appendixes, and there are some additional optional features listed in Section 4.10.4. The optional features are not strictly required for policy creation and deployment but listed to help make the developed tool more useful and better suiting to meet the set goal.

4.10.3 Foundations for building the semi-automated process

Pre-requisites

Before starting with the deployment of Application Control solutions, make sure to consider the expectations to IT maturity described in section 4.2.1 of 'Suitable target organizations' and the general description of 'Application Control for Windows' in section 4.5.

Decision on the approach - allowlist or blocklist

The next most important decision is to consider if the fundamental approach is to block anything specifically allowed - the allowlist approach. Or to allow anything specifically blocked - the blocklisting approach.

Next, the most crucial decision is to choose the fundamental approach, either allowlisting or blocklisting. The difference is explained in 4.5.

While blocklisting is less demanding on the effort and IT maturity, it also has a lesser effect. It is likely to prevent automated attacks and leave plenty of ways for advanced attackers to circumvent the restrictions.

Management of rules

This is the first step in building automation. The approach is demonstrated as proof-of-concept with script 'LOLBAS-filepaths.py' in Appendix 5.1.4, where data from an external source is collected and parsed into a format that can be used as a single point of management for multiple security features with a single goal.

The rules management does not necessarily have to be implemented with CSV files. The management could be built on a simple database with a Web interface with little effort.

The importance is that the rules can then be easily moved from machine-level restrictions by WDAC to the user or group-based rules by AppLocker, and edge-cases and exceptions can be handled in the same environment. This allows for logging and versioning, either via Git when the back-end is built on files or via a dedicated solution for a custom application.

Analysis

Before creating the policies using the input from the rule management solution, an analysis should be performed to gather data on the daily administration procedures and the trusted tools used by users and administrators. These files should not be blocked on the machine

level but based on users or groups.

The example rules and approaches described in section 4.8, 'Implementing the rules' can be used for basis, created based on the design descriptions in sections 4.5.2, 4.5.3, 4.5.4, 4.6.

The decisions should be reflected in the rule management solution. Together with comments for files excluded from restrictions altogether, if necessary. For exclusions, it is recommended to include reasoning and revision date.

Creation of policies

After creating the new iteration of analysis and rule updates, the rules can be automatically converted to Windows Defender Application Control and AppLocker policies. The automation can be built similarly to the proof-of-concept scripts provided, *LOLBAS_WDAC-policy.py* in Appendix 5.1.4 and *LOLBAS_AppLocker-policy.py* in Appendix 5.1.4, for WDAC and AppLocker policies respectively.

The scripts can be merged and improved as necessary for the processes and procedures in the specific organization. There is a need to utilize some built-in Powershell Cmdlets too. Specific process is described in sections 4.8.2 and 4.8.3.

This step should be fully automatable.

Deployment of policies

In the first iterations of policy creation and deployment, it is strongly suggested to start with deploying policies set to audit only - logging and monitoring the current situation with new policies and not restricting the execution of binaries and scripts.

This provides a good ground for testing and improving the rules without the risk of impacting the everyday work of users and administrators.

The automation capabilities are dependent on the specific endpoint management approach in use - Group Policy, Microsoft Intune, or any alternative solutions. In most cases, likely, system administrators can simply push replacement policy files for AppLocker and WDAC without changing the enforcing policies every time.

Collecting logs

It is crucial to collect logs from endpoints to a central location to allow for data-driven analysis and improvement of policy deployment iterations. Most log collection or Security Incident and Event Monitoring (SIEM) solutions should be sufficient.

The Windows event IDs and locations to track are described in 4.8.2 and 4.8.3.

Repeat the last four steps until satisfied with the results.

Transitioning to *Enforce* mode

Before deploying the policies in enforcement mode, it is important to notify stakeholders of the plan and inform them of the support procedures and channels in case the policies are too restrictive or have other unintended effects.

The steps to configure enforcement of policies are described in 4.8.2 and 4.8.3 for WDAC and AppLocker, respectively.

Log analysis

Although the policies are deployed, it is still a good practice to keep collecting the logs and to regularly review the logs. There might be some issues not evident to end-users, for example, restricting software updates.

Moreover, the logs also might include indications of attacks in the environment. These events should be forwarded or made available for the security team or CSIRT.

Repeat analysis and policy updating steps as necessary.

4.10.4 Additional/optional features

The following features could be considered as not required to create and deploy the defensive policies, but incorporating the functionalities would make the tool more sustainable in the long run and simplify the management.

Although the primary goal is covered as a methodology in Section 4.10.3 - to create the most useful solution when implementing, there are additional features to consider:

- Active Directory integration to automatically collect SIDs of users or groups to be

used in policy files

- The user interface should store the previously made policy configurations and merge them with newly added binaries from the data sources
- The management tool should allow to add comments and revision dates to excluded binaries
- The tool should allow to automatically generate and include rules from golden-image machines to reduce the effort of allowlisting known applications
- The management tool should incorporate versioning and history of changes

5. Summary

The outcome of this study is a methodology consisting of nine steps, with each step further described in the body of the work, on how to create a process and develop a tool to build and manage a comprehensive set of policies to defend against Living off the Land attacks in a semi-automated manner.

The document includes technical information on the trusted binaries in the system and resources listing the potentially dangerous files in this set. The Windows operating system features usable for defending against the attacks are analyzed, and descriptions of capabilities, limitations and other important considerations for implementation are included.

The study presents methods and proof-of-concept scripts to display the potential of a semi-automatic way of managing multiple security features from a single point in a simplified way. The designed approach was implemented and tested with eight test cases, assessing the policies and demonstrating the processes and means for collecting data for an iterative improvement routine.

The automated process provides quicker deployment cycles and simplified management, as both WDAC and AppLocker rules can be managed from single locations. Together with documenting binaries not included in either policies or which edge-cases need alternative approaches, like the use of Access Control List.

The foundations for building the semi-automated management and deployment procedure are provided to readers as a distinct, concise section, referring to other sections for the details. Enabling organizations to follow the steps as the basis for building custom solution best fitting their requirements, or to be the foundation for efficient management solution development.

While the approaches described in the study are useful to close down the potential avenues for code execution by limiting Living off the Land binaries in the system, by blocking the initial code execution or breaking the execution chain down the line for attacks using the techniques at hand, the approach should not be used as a single line of defense, but as

additional measure to other cyber defense practices.

5.1 Future work

5.1.1 Development of a management tool with user-friendly interface based on the method proposed

Development of the tool to automatically and periodically import the data sources for potentially dangerous binaries, parsing into configuration file and generating the policy files, as shown in the method proposed.

Implementing the tool with at least some of the optional features listed in the paper to create a tool suitable for continuous use and policy management.

5.1.2 Evaluating the described Living off the Land defenses in enterprise environment

As demonstrated in the study, building and using automation for managing the Windows Defender Application Control and AppLocker, including external data sources like 'Microsoft recommended block rules'[36] and the LOLBAS project[19] to enhance the policies.

Provide real-life experience and input for improving the methods presented in the study.

Create statistical comparison to discover trends in the amounts of incidents before and after deployment in enforcing mode.

5.1.3 Analysis of recommended drivers block rules by Microsoft

Analysis of Microsoft provided 'Microsoft recommended driver block rules'[73] to compare with other similar resources regarding potentially dangerous drivers. Demonstrate how to implement the restrictive rules to defend against such drivers and analyze the current situation on IT infrastructure to avoid blocking drivers in use.

5.1.4 Using Just Enough Administration for locking down Powershell in Workstations

Microsoft has impressive functionality called Just Enough Administration (JEA)[74], which provides a secure environment for administrators to manage remote endpoints with limited and controlled sets of Powershell Cmdlets and arguments available.

If Microsoft enabled using the JEA feature to lock down the local Powershell usage for standard users, it would become a tool with great potential in restricting Powershell as a tool for Living off the Land attacks.

Bibliography

- [1] Barr-Smith Frederick; Ugarte-Pedrero Xabier; Graziano Mariano; Spolaor Riccardo; Martinovic Ivan. *Survivalism: Systematic Analysis of Windows Malware Living-Off-The-Land*. [Accessed: 30-03-2022]. URL: <https://ieeexplore.ieee.org/abstract/document/9519480>.
- [2] Microsoft. *Windows Defender Application Control (WDAC) example base policies*. [Accessed: 06-04-2022]. URL: <https://docs.microsoft.com/en-us/windows/security/threat-protection/windows-defender-application-control/example-wdac-base-policies>.
- [3] Oddovar Moe. *The LOLBAS Project - MITRE ATT&CK mapping*. [Accessed: 10-05-2022]. URL: https://mitre-attack.github.io/attack-navigator/#layerURL=https://lolbas-project.github.io/mitre_attack_navigator_layer.json.
- [4] Symantec. *Internet Security Threat Report - Living off the land and fileless attack techniques*. [Accessed: 31-03-2022]. URL: <https://docs.broadcom.com/doc/istr-24-2019-en>.
- [5] Kwan Lin, Bob Rudis, Katie Wilbur, Wade Woolwine. *Quarterly Threat Report: Q3 2019*. [Accessed: 25-04-2022]. URL: <https://www.rapid7.com/research/report/2019-q3-threat-report/>.
- [6] Microsoft. *Fileless threats*. [Accessed: 31-03-2022]. URL: <https://docs.microsoft.com/en-us/microsoft-365/security/intelligence/fileless-threats?view=o365-worldwide>.
- [7] Travis Gregory; Balas Ed; Ripley David; Wallac Steven. *Analysis of the “SQL Slammer” worm and its effects on Indiana University and related institutions*. [Accessed: 31-03-2022]. URL: <https://ivanlef0u.fr/repo/madchat/vxdevl/papers/analysis/SLAMMER.pdf>.
- [8] MITRE. *MITRE ATT&CK - Event Triggered Execution: Image File Execution Options Injection*. [Accessed: 31-03-2022]. URL: <https://attack.mitre.org/techniques/T1546/012/>.
- [9] Oddovar Moe. *Github - Living Off The Land Binaries And Scripts*. [Accessed: 25-03-2022]. URL: <https://github.com/LOLBAS-Project/LOLBAS>.

- [10] David Brown. *Preventing Living off the Land Attacks*. [Accessed: 25-04-2022]. URL: <https://www.giac.org/paper/gpen/8849/preventing-living-land-attacks/140526>.
- [11] Symantec. *The Increased Use Of Powershell In Attacks*. [Accessed: 21-04-2022]. URL: <https://docs.broadcom.com/doc/increased-use-of-powershell-in-attacks-16-en>.
- [12] Symantec. *Internet Security Threat Report - Volume 24*. [Accessed: 11-05-2022]. URL: <https://docs.broadcom.com/doc/istr-living-off-the-land-and-fileless-attack-techniques-en>.
- [13] Symantec. *The Ransomware Threat Landscape: What to Expect in 2022*. [Accessed: 11-05-2022]. URL: https://www.symantec.broadcom.com/hubfs/SED/SED_Threat_Hunter_Reports_Alerts/SED_FY22Q2_SES_Ransomware-Threat-Landscape_WP.pdf.
- [14] CrowdStrike. *2022 Global Threat Report*. [Accessed: 11-05-2022]. URL: <https://go.crowdstrike.com/rs/281-OBQ-266/images/Report2022GTR.pdf>.
- [15] MITRE. *MITRE ATT&CK - System Binary Proxy Execution*. [Accessed: 11-05-2022]. URL: <https://attack.mitre.org/techniques/T1218/>.
- [16] Red Canary. *2021 Threat Detection Report*. [Accessed: 11-05-2022]. URL: <https://resource.redcanary.com/rs/003-YRU-314/images/2021-Threat-Detection-Report.pdf>.
- [17] Rohan Durve and Ahmed Bouridane. "Windows 10 security hardening using device guard whitelisting and Applocker blacklisting". In: *2017 Seventh International Conference on Emerging Security Technologies (EST)*. 2017, pp. 56–61. DOI: 10.1109/EST.2017.8090399.
- [18] Aaron Beuhring and Kyle Salous. "Beyond Blacklisting: Cyberdefense in the Era of Advanced Persistent Threats". In: *IEEE Security Privacy* 12.5 (2014), pp. 90–93. DOI: 10.1109/MSP.2014.86.
- [19] Oddovar Moe. *Living Off The Land Binaries And Scripts*. [Accessed: 18-11-2021]. URL: <https://lolbas-project.github.io>.
- [20] NirSoft. *NirSoft*. [Accessed: 06-04-2022]. URL: <https://www.nirsoft.net>.
- [21] Benjamin 'gentilkiwi' Delpy. *mimikatz*. [Accessed: 06-04-2022]. URL: <https://blog.gentilkiwi.com/mimikatz>.
- [22] CISA. *CISA - Publicly Available Tools Seen in Cyber Incidents Worldwide*. [Accessed: 06-04-2022]. URL: <https://www.cisa.gov/uscert/ncas/alerts/AA18-284A>.

- [23] Microsoft. *Understand Windows Defender Application Control policy design decisions*. [Accessed: 04-04-2022]. URL: <https://docs.microsoft.com/en-gb/windows/security/threat-protection/windows-defender-application-control/understand-windows-defender-application-control-policy-design-decisions>.
- [24] Microsoft. *Get-AuthenticodeSignature*. [Accessed: 27-03-2022]. URL: <https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.security/get-authenticodesignature?view=powershell-7.2>.
- [25] Microsoft. *Signature.IsOSBinary Property*. [Accessed: 27-03-2022]. URL: <https://docs.microsoft.com/en-us/dotnet/api/system.management.automation.signature.isosbinary?view=powershellsdk-7.0.0#system-management-automation-signature-isosbinary>.
- [26] Microsoft. *Signature Class*. [Accessed: 27-03-2022]. URL: <https://docs.microsoft.com/en-us/dotnet/api/system.management.automation.signature?view=powershellsdk-7.0.0>.
- [27] Microsoft. *Application Control for Windows*. [Accessed: 06-04-2022]. URL: <https://docs.microsoft.com/en-us/windows/security/threat-protection/windows-defender-application-control/windows-defender-application-control>.
- [28] Microsoft. *Executable rules in AppLocker*. [Accessed: 06-04-2022]. URL: <https://docs.microsoft.com/en-us/windows/security/threat-protection/windows-defender-application-control/applocker/executable-rules-in-applocker>.
- [29] Microsoft. *Windows Defender Application Control design guide*. [Accessed: 06-04-2022]. URL: <https://docs.microsoft.com/en-us/windows/security/threat-protection/windows-defender-application-control/windows-defender-application-control-design-guide>.
- [30] Microsoft. *Requirements to use AppLocker*. [Accessed: 07-04-2022]. URL: <https://docs.microsoft.com/en-us/windows/security/threat-protection/windows-defender-application-control/applocker/requirements-to-use-applocker>.
- [31] Microsoft. *Windows Defender Application Control and AppLocker feature availability*. [Accessed: 06-04-2022]. URL: <https://docs.microsoft.com/en-gb/windows/security/threat-protection/windows-defender-application-control/feature-availability>.

- [32] Microsoft. *New-CIPolicy*. [Accessed: 25-04-2022]. URL: <https://docs.microsoft.com/en-us/powershell/module/configci/new-cipolicy?view=windowsserver2022-ps>.
- [33] Microsoft. *Windows Defender Application Control and AppLocker Overview*. [Accessed: 06-04-2022]. URL: <https://docs.microsoft.com/en-gb/windows/security/threat-protection/windows-defender-application-control/wdac-and-applocker-overview>.
- [34] Microsoft. *Understand Windows Defender Application Control (WDAC) policy rules and file rules*. [Accessed: 06-04-2022]. URL: <https://docs.microsoft.com/en-us/windows/security/threat-protection/windows-defender-application-control/select-types-of-rules-to-create>.
- [35] TROY MARTIN. *Device Guard: The Theory*. [Accessed: 06-04-2022]. URL: <https://www.1e.com/news-insights/blogs/device-guard-theory/>.
- [36] Microsoft. *Microsoft recommended block rules*. [Accessed: 06-04-2022]. URL: <https://docs.microsoft.com/en-us/windows/security/threat-protection/windows-defender-application-control/microsoft-recommended-block-rules>.
- [37] Microsoft. *Select the types of rules to create*. [Accessed: 07-04-2022]. URL: <https://docs.microsoft.com/en-us/windows/security/threat-protection/windows-defender-application-control/applocker/select-types-of-rules-to-create>.
- [38] Microsoft. *Understanding the publisher rule condition in AppLocker*. [Accessed: 07-04-2022]. URL: <https://docs.microsoft.com/en-us/windows/security/threat-protection/windows-defender-application-control/applocker/understanding-the-publisher-rule-condition-in-applocker>.
- [39] Microsoft. *Understanding the path rule condition in AppLocker*. [Accessed: 07-04-2022]. URL: <https://docs.microsoft.com/en-us/windows/security/threat-protection/windows-defender-application-control/applocker/understanding-the-path-rule-condition-in-applocker>.
- [40] Microsoft. *Understanding the file hash rule condition in AppLocker*. [Accessed: 07-04-2022]. URL: <https://docs.microsoft.com/en-us/windows/security/threat-protection/windows-defender-application-control/applocker/understanding-the-file-hash-rule-condition-in-applocker>.

- control / applocker / understanding - the - file - hash - rule - condition-in-applocker.
- [41] Microsoft. *Understanding AppLocker allow and deny actions on rules*. [Accessed: 18-04-2022]. URL: <https://docs.microsoft.com/en-us/windows/security/threat-protection/windows-defender-application-control/applocker/understanding-applocker-allow-and-deny-actions-on-rules>.
 - [42] John Lambert (@JohnLaTwC) Jimmy (@bohops) Daniel Bohannon (@danielbohannon). *Living Off The Land Binaries And Scripts - Manage-bde.wsf*. [Accessed: 07-04-2022]. URL: <https://lolbas-project.github.io/lolbas/Scripts/Manage-bde/>.
 - [43] Microsoft. *Access Control List*. [Accessed: 16-04-2022]. URL: <https://docs.microsoft.com/en-us/windows-hardware/drivers/ifs/access-control-list>.
 - [44] Microsoft. *Security descriptors in file systems*. [Accessed: 16-04-2022]. URL: <https://docs.microsoft.com/en-us/windows-hardware/drivers/ifs/security-descriptors>.
 - [45] Microsoft. *Access Control Overview*. [Accessed: 16-04-2022]. URL: <https://docs.microsoft.com/en-us/windows/security/identity-protection/access-control/access-control>.
 - [46] Microsoft. *manage-bde*. [Accessed: 16-04-2022]. URL: <https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/manage-bde>.
 - [47] MITRE. *MITRE ATT&CK - Signed Script Proxy Execution*. [Accessed: 16-04-2022]. URL: <https://attack.mitre.org/techniques/T1216/>.
 - [48] Microsoft. *DACLs and ACEs*. [Accessed: 16-04-2022]. URL: <https://docs.microsoft.com/en-us/windows/win32/secauthz/dacls-and-aces>.
 - [49] Microsoft. *Order of ACEs in a DACL*. [Accessed: 16-04-2022]. URL: <https://docs.microsoft.com/en-us/windows/win32/secauthz/order-of-aces-in-a-dacl>.
 - [50] Zhenyuan Li et al. “Effective and Light-Weight Deobfuscation and Semantic-Aware Attack Detection for PowerShell Scripts”. In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. CCS ’19. London, United Kingdom: Association for Computing Machinery, 2019, pp. 1831–1847. ISBN: 9781450367479. DOI: 10.1145/3319535.3363187. URL: <https://doi.org/10.1145/3319535.3363187>.

- [51] D. Ugarte et al. *PowerDrive: Accurate De-obfuscation and Analysis of PowerShell Malware*. [Accessed: 25-04-2022]. URL: https://link.springer.com/chapter/10.1007/978-3-030-22038-9_12.
- [52] MITRE ATT&CK. *Command and Scripting Interpreter: PowerShell*. [Accessed: 21-04-2022]. URL: <https://attack.mitre.org/techniques/T1059/001/>.
- [53] Tjada Nelson and Houssain Kettani. “Open Source PowerShell-Written Post Exploitation Frameworks Used by Cyber Espionage Groups”. In: *2020 3rd International Conference on Information and Computer Technologies (ICICT)*. 2020, pp. 451–456. DOI: 10.1109/ICICT50521.2020.00078.
- [54] Microsoft. *about_Language_Modes*. [Accessed: 21-04-2022]. URL: https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about%5C_language%5C_modes%5C?view=powershell-7.2.
- [55] Microsoft Paul Higinbotham PowerShell Team. *PowerShell Constrained Language Mode*. [Accessed: 21-04-2022]. URL: <https://devblogs.microsoft.com/powershell/powershell-constrained-language-mode/>.
- [56] Microsoft. *Merge-CIPolicy*. [Accessed: 17-04-2022]. URL: <https://docs.microsoft.com/en-us/powershell/module/configci/merge-cipolicy?view=windowsserver2022-ps>.
- [57] Microsoft. *Merge Windows Defender Application Control (WDAC) policies*. [Accessed: 17-04-2022]. URL: <https://docs.microsoft.com/en-us/windows/security/threat-protection/windows-defender-application-control/merge-windows-defender-application-control-policies>.
- [58] Microsoft. *ConvertFrom-CIPolicy*. [Accessed: 17-04-2022]. URL: <https://docs.microsoft.com/en-us/powershell/module/configci/convertfrom-cipolicy?view=windowsserver2022-ps>.
- [59] Microsoft. *Deploy Windows Defender Application Control policies by using Group Policy*. [Accessed: 17-04-2022]. URL: <https://docs.microsoft.com/en-us/windows/security/threat-protection/windows-defender-application-control/deploy-windows-defender-application-control-policies-using-group-policy>.
- [60] Microsoft. *Understanding Application Control events*. [Accessed: 22-04-2022]. URL: <https://docs.microsoft.com/en-us/windows/security/threat-protection/windows-defender-application-control/event-id-explanations>.

- [61] Holger Rünkaru. *LOLBAS-filepaths_exampleoutput.csv*. [Accessed: 18-04-2022]. URL: https://github.com/hrunkaru/LOTLDefence/blob/main/LOLBAS-filepaths%5C_exampleoutput.csv.
- [62] Microsoft. *Configure the Application Identity service*. [Accessed: 19-04-2022]. URL: <https://docs.microsoft.com/en-us/windows/security/threat-protection/windows-defender-application-control/applocker/configure-the-application-identity-service>.
- [63] Oddvar Moe (@oddvarmoe) Casey Smith (@subtee). *Living Off The Land Binaries And Scripts - Mshta.exe*. [Accessed: 19-04-2022]. URL: <https://lolbas-project.github.io/lolbas/Binaries/Mshta/>.
- [64] MITRE ATT&CK. *Signed Binary Proxy Execution: Mshta*. [Accessed: 19-04-2022]. URL: <https://attack.mitre.org/techniques/T1218/005/>.
- [65] Keith Mccammon. *Microsoft HTML Application (HTA) Abuse, Part Deux*. [Accessed: 19-04-2022]. URL: <https://redcanary.com/blog/microsoft-html-application-hta-abuse-part-deux/>.
- [66] MITRE ATT&CK. *Scheduled Task/Job: Scheduled Task*. [Accessed: 19-04-2022]. URL: <https://attack.mitre.org/techniques/T1053/005/>.
- [67] N/A. *Living Off The Land Binaries And Scripts - Mshta.exe*. [Accessed: 19-04-2022]. URL: <https://lolbas-project.github.io/lolbas/Binaries/Schtasks/>.
- [68] GitHub - Pester. *Pester*. [Accessed: 19-04-2022]. URL: <https://github.com/pester/Pester>.
- [69] Emin Atac (@p0w3rsh3ll). *Living Off The Land Binaries And Scripts - Pester.bat*. [Accessed: 19-04-2022]. URL: <https://lolbas-project.github.io/lolbas/Scripts/pester/>.
- [70] MITRE ATT&CK. *Signed Script Proxy Execution*. [Accessed: 19-04-2022]. URL: <https://attack.mitre.org/techniques/T1216/>.
- [71] Jimmy; Fabrizio; Moriarty; Nick Carr. *Living Off The Land Binaries And Scripts - Advpack.dll*. [Accessed: 20-04-2022]. URL: <https://lolbas-project.github.io/lolbas/Libraries/Advpack/>.
- [72] MITRE ATT&CK. *Signed Script Proxy Execution: Rundll32*. [Accessed: 20-04-2022]. URL: <https://attack.mitre.org/techniques/T1218/011/>.
- [73] Microsoft. *Microsoft recommended driver block rules*. [Accessed: 25-04-2022]. URL: <https://docs.microsoft.com/en-us/windows/security/threat-protection/windows-defender-application-control/microsoft-recommended-driver-block-rules>.

[74] Microsoft. *Just Enough Administration*. [Accessed: 25-04-2022]. URL: <https://docs.microsoft.com/en-us/powershell/scripting/learn/remoting/jea/overview?view=powershell-7.2>.

Appendices

Appendix 1 - LOLBAS-filepaths.py

```
import os
from pathlib import Path
import yaml
# pip3 install pyyaml to install module 'yaml'
import csv
import argparse
import time

def cleanYAMLlastline(data):
    if (data[-1:][0] == "---\n"):
        newdata = data[:-1]
    return newdata

def createFilepaths(pathToYamls):
    filepaths = []
    subdir_list = os.listdir(pathToYamls)
    for dir in subdir_list:
        filenames = os.listdir(Path(pathToYamls, dir))
        for filename in filenames:
            filepaths.append(str((Path(dir, filename))))
    return filepaths

def createCSVsummary(filepaths_list, add_block_policy):
    with open (pathToOutputCSV, 'w') as csv_out:

        #Create CSV header
        writer = csv.writer(csv_out)
        if(add_block_policy):
            writer.writerow(["Category", "Name", "Description",
```



```

        "Path", "Block_in_WDAC/AppLocker", "Privileges"])
else:
    writer.writerow(["Category", "Name", "Description",
        "Path", "Privileges"])

for file in filepath_list:
    splitname = file.split(".")

    with open(Path(pathToYAMLs, file), 'r') as yml_in:
        lines = yml_in.readlines()
        # Clean YAML files from appended '---' when no
        # item is following
        yml_data = cleanYAMLlastline(lines)
        data = yaml.load(''.join(yml_data), Loader=yaml
            .FullLoader)

        recordtype = file.split("/")[-2]

        common_values = []
        common_values.append(recordtype)
        common_values.append(data['Name'] if ('Name' in
            data) else "N/A")
        common_values.append(data['Description'] if all
            (['Description' in data, len(str(data['
                Description'])) > 1]) else "N/A")

    try:
        for fpath in data['Full_Path']:
            write_values = []
            write_values.extend(common_values)
            if fpath['Path']:
                if len(str(fpath['Path'])) > 1:
                    write_values.append(fpath['Path'
                        ])
                else:
                    write_values.append("N/A")
            else:
                write_values.append("N/A")
            if(add_block_policy):
                write_values.append("")

```

```

        write_values.append(data['Commands'][0][
            'Privileges'] if ('Privileges' in
            data['Commands'][0]) else "N/A")
        writer.writerow(write_values)
except :
    write_values = []
    write_values.extend(common_values)
    write_values.append("N/A")
    if(add_block_policy):
        write_values.append("")
    write_values.append(data['Commands'][0]['
        Privileges'] if ('Privileges' in data['
        Commands'][0]) else "N/A")
    writer.writerow(write_values)

if __name__ == "__main__":
    # Get and parse arguments
    parser = argparse.ArgumentParser(description="Parser_script_
        to_collect_filepaths_from_LOLBAS_project_YAML_files._\
        nLOLBAS_Project:_https://github.com/LOLBAS-Project/LOLBAS
        _\nParser_script_from:_https://github.com/hrunkaru/
        LOTLDefence",
                                     formatter_class=argparse.
                                     ArgumentDefaultsHelpFormatter
                                     )
    parser.add_argument("-o", "--output", help="Output_CSV_file_
        path_(default_is_current_folder)")
    parser.add_argument("-s", "--src", help="Path_to_local_copy_
        of_LOLBAS_project", required=True)
    parser.add_argument("-p", "--policyprep", action='store_true
        ', help="Add_column_to_CSV_to_mark_rules_for_blocking_in_
        policy_creator_scripts.")
    args = parser.parse_args()

    # path to local copy of LOLBAS project files from this
    # project:
    # https://github.com/LOLBAS-Project/LOLBAS

```

```

pathToLOLBAS = args.src
# path to yml files in the LOLBAS project
pathToYAMLs = Path(pathToLOLBAS, "yaml")

# output file name and path
timestr = time.strftime("_%Y%m%d_%H%M%S")
if(args.output):
    pathToOutputCSV = args.output
else:
    pathToOutputCSV = 'LOLBAS_filepaths' + timestr + '.csv'

# create list of filepaths to iterate
filepaths = createFilepaths(pathToYAMLs)

# create CSV summary of LOLBAS yaml files
if(args.policyprep):
    createCSVsummary(filepaths, True)
else:
    createCSVsummary(filepaths)

# print output when done
print ("Parsing_done._Find_the_output_file_at:_\n" +
        pathToOutputCSV)

```

Appendix 2 - Get-LOTLInfo.ps1

```
param ($pathToPathsCSV = "LOLBAS_filepaths.csv", $outfile = "
    LOTLInfo_$(Get-Date -UFormat "%Y-%m-%d_%H-%M-%S").csv")

$data = import-csv $pathToPathsCSV
$outputdata = @()
$countTestPath = 0

write-debug "Total items found from input CSV file: $($data |
    Measure-Object | Select-Object -ExpandProperty count)"

# filtering by type
# Include both OSBinaries and OtherMSBinaries, exclude
    OSLibraries and OSScripts
#write-debug "Total items after filtering by type: $($data |
    Where-Object Category -like "*Binaries" | Measure-Object |
    Select-Object -ExpandProperty count)"
#$data | Where-Object Category -like "*Binaries" | ForEach-
    Object {

# no filtering
$data | ForEach-Object {
#
    if (Test-Path -Path $_.Path){
        $countTestPath += 1

        $item = $(Get-AuthenticodeSignature $_.Path |
            Select-Object -Property * -ExpandProperty
                SignerCertificate |
            Select-Object statusmessage, signaturetype, isosbinary,
                issuer, subject, Name, Path
        )
        $item.Name = $_.Name
        $item.Path = $_.Path
        $outputdata += $item
    }
}
```

```
}
else {
    $item = $_ | Select-Object statusmessage, signaturetype,
        isosbinary, issuer, subject, Name, Path
    $item.StatusMessage = "File not found"
    $outputdata += $item
}
}

write-debug "Total existing paths in the system: $countTestPath"
$outputdata | Export-Csv -Path $outfile -NoTypeInfoation
write-debug "Total items in output data: $($outputdata | Measure
    -Object | Select-Object -expandproperty count)"

write-host "All done, find the output at: $outfile"
```

Appendix 3 - LOLBAS_WDAC-policy.py

```
import csv
import argparse
import time

def createWDACPolicy(csv_in):

    xml_template = """\
<?xml version="1.0" encoding="utf-8" ?>
<SiPolicy xmlns="urn:schemas-microsoft-com:sipolicy">
  <FileRules>
    {FRules}
  </FileRules>
  <SigningScenarios>
    <SigningScenario Value="12" ID="ID_SIGNINGSCENARIO_WINDOWS"
      FriendlyName="User Mode Signing Scenarios">
      <ProductSigners>
        <FileRulesRef>
          {FRulesRef}
        </FileRulesRef>
      </ProductSigners>
    </SigningScenario>
  </SigningScenarios>
  <UpdatePolicySigners />
  <CiSigners />
  <HvciOptions>0</HvciOptions>
</SiPolicy>\
    """

    FileRules = ''
    FileRulesRef = ''

    with open(csv_in) as input_csv:
        data = csv.reader(input_csv)
        IDcount = 1
        for row in data:
```

```

if(row[4] == "WDAC"):

    #IDstr = 'ID_DENY_LOLBAS_' + str(IDcount) + '_'
        + row[0]
    IDstr = 'ID_DENY_LOLBAS_' + row[0].upper() + '_'
        + str(IDcount)
    IDcount += 1
    FriendlyName = row[1]
    FilePath = row[3]

    FileRules += f'\t\t<Deny_ID="{IDstr}"_
        FriendlyName="{FriendlyName}"_FilePath="{
        FilePath}"_>\n'

    FileRulesRef += f'\t\t<FileRuleRef_RuleID="{
        IDstr}"_>\n'

output = xml_template.format(FRules=FileRules, FRulesRef=
    FileRulesRef)
return output

if __name__ == "__main__":
    # Get and parse arguments
    parser = argparse.ArgumentParser(description="Helper_script_
        to_create_policy_XML_for_Windows_Defender_Application_
        Control_from_CSV_input,_script_is_from:_https://github.
        com/hrunkaru/LOTLDefence",
        formatter_class=argparse.
            ArgumentDefaultsHelpFormatter
        )
    parser.add_argument("-o", "--wdacoutput", help="Output_XML_
        file_path_for_WDAC_policy_(default_is_current_folder)")
    parser.add_argument("-s", "--src", help="Path_to_CSV_output_
        from_LOLBAS-filepaths.py_script_(-p_switch),_where_
        required_rows_are_marked_with_AppLocker_or_WDAC_manually"
        , required=True)
    args = parser.parse_args()

```

```

# Paths
## Input
CSV_in = args.src

## Timestring to use in names
timestr = time.strftime("_%Y%m%d_%H%M%S")

## WDAC Output file
if(args.wdacoutput):
    WDACOutput = 'args.wdacoutput'
else:
    WDACOutput = 'WDAC_policy' + timestr + '.xml'

# Create data to output
output_data = createWDACPolicy(CSV_in)

# Save output to file
with open (WDACOutput, 'w') as policy_out:
    policy_out.write(output_data)
    print ("All_done, find_output_file_at:_" + WDACOutput)

```


Appendix 4 - AppLocker-example.xml

```
<AppLockerPolicy Version="1">
  <RuleCollection Type="Appx" EnforcementMode="AuditOnly" />
  <RuleCollection Type="Dll" EnforcementMode="AuditOnly">
    <FilePublisherRule Id="e2089c56-a41a-49a0-9631-c27867e73003"
      Name="Signed_by_O=MICROSOFT_CORPORATION,_L=REDMOND,_S=
      WASHINGTON,_C=US" Description="Block_Microsoft_signed_
      file_execution_outside_of_allowed_locations_(exceptions)"
      UserOrGroupSid="S
      -1-5-21-2508682811-3451744959-1246442822-1112" Action="
      Deny">
      <Conditions>
        <FilePublisherCondition PublisherName="O=MICROSOFT_
          CORPORATION,_L=REDMOND,_S=WASHINGTON,_C=US"
          ProductName="*" BinaryName="*">
          <BinaryVersionRange LowSection="*" HighSection="*" />
        </FilePublisherCondition>
      </Conditions>
      <Exceptions>
        <FilePathCondition Path="%OSDRIVE%\ProgramData\*" />
        <FilePathCondition Path="%PROGRAMFILES%\*" />
        <FilePathCondition Path="%WINDIR%\*" />
      </Exceptions>
    </FilePublisherRule>
    <FilePathRule Id="3737732c-99b7-41d4-9037-9cddf0de0d0" Name
      ="(Default_Rule)_All_DLLs_located_in_the_Program_Files_
      folder" Description="Allows_members_of_the_Everyone_group
      to_load_DLLs_that_are_located_in_the_Program_Files_
      folder." UserOrGroupSid="S-1-1-0" Action="Allow">
      <Conditions>
        <FilePathCondition Path="%PROGRAMFILES%\*" />
      </Conditions>
    </FilePathRule>
    <FilePathRule Id="bac4b0bf-6f1b-40e8-8627-8545fa89c8b6" Name
      ="(Default_Rule)_Microsoft_Windows_DLLs" Description="
      Allows_members_of_the_Everyone_group_to_load_DLLs_located
```

```

        _in_the_Windows_folder." UserOrGroupSid="S-1-1-0" Action=
        "Allow">
    <Conditions>
        <FilePathCondition Path="%WINDIR%\*" />
    </Conditions>
</FilePathRule>
<FilePathRule Id="fe64f59f-6fca-45e5-a731-0f6715327c38" Name
    ="(Default_Rule)_All_DLLs" Description="Allows_members_of
    _the_local_Administrators_group_to_load_all_DLLs."
    UserOrGroupSid="S-1-5-32-544" Action="Allow">
    <Conditions>
        <FilePathCondition Path="*" />
    </Conditions>
</FilePathRule>
</RuleCollection>
<RuleCollection Type="Exe" EnforcementMode="AuditOnly">
    <FilePublisherRule Id="e21e4b2c-665d-43ab-a7b7-9e18cfd60c83"
        Name="Signed_by_O=MICROSOFT_CORPORATION,_L=REDMOND,_S=
        WASHINGTON,_C=US" Description="Block_Microsoft_signed_
        file_execution_outside_of_allowed_locations_(exceptions)"
        UserOrGroupSid="S
        -1-5-21-2508682811-3451744959-1246442822-1112" Action="
        Deny">
    <Conditions>
        <FilePublisherCondition PublisherName="O=MICROSOFT_
            CORPORATION,_L=REDMOND,_S=WASHINGTON,_C=US"
            ProductName="*" BinaryName="*">
            <BinaryVersionRange LowSection="*" HighSection="*" />
        </FilePublisherCondition>
    </Conditions>
    <Exceptions>
        <FilePathCondition Path="%OSDRIVE%\ProgramData\*" />
        <FilePathCondition Path="%PROGRAMFILES%\*" />
        <FilePathCondition Path="%WINDIR%\*" />
    </Exceptions>
</FilePublisherRule>
<FilePathRule Id="921cc481-6e17-4653-8f75-050b80acca20" Name
    ="(Default_Rule)_All_files_located_in_the_Program_Files_
    folder" Description="Allows_members_of_the_Everyone_group
    _to_run_applications_that_are_located_in_the_Program_
    Files_folder." UserOrGroupSid="S-1-1-0" Action="Allow">

```

```

    <Conditions>
      <FilePathCondition Path="%PROGRAMFILES%\*" />
    </Conditions>
  </FilePathRule>
  <FilePathRule Id="a61c8b2c-a319-4cd0-9690-d2177cad7b51" Name
    ="(Default_Rule)_All_files_located_in_the_Windows_folder"
    Description="Allows_members_of_the_Everyone_group_to_run
    _applications_that_are_located_in_the_Windows_folder."
    UserOrGroupSid="S-1-1-0" Action="Allow">
    <Conditions>
      <FilePathCondition Path="%WINDIR%\*" />
    </Conditions>
  </FilePathRule>
  <FilePathRule Id="fd686d83-a829-4351-8ff4-27c7de5755d2" Name
    ="(Default_Rule)_All_files" Description="Allows_members_
    of_the_local_Administrators_group_to_run_all_applications
    ." UserOrGroupSid="S-1-5-32-544" Action="Allow">
    <Conditions>
      <FilePathCondition Path="*" />
    </Conditions>
  </FilePathRule>
</RuleCollection>
<RuleCollection Type="Msi" EnforcementMode="AuditOnly" />
<RuleCollection Type="Script" EnforcementMode="AuditOnly">
  <FilePublisherRule Id="fcaea78e-dba2-4120-b9f2-3b2cc9934586"
    Name="Signed_by_O=MICROSOFT_CORPORATION,_L=REDMOND,_S=
    WASHINGTON,_C=US" Description="Block_Microsoft_signed_
    file_execution_outside_of_allowed_locations_(exceptions)"
    UserOrGroupSid="S
    -1-5-21-2508682811-3451744959-1246442822-1112" Action="
    Deny">
    <Conditions>
      <FilePublisherCondition PublisherName="O=MICROSOFT_
        CORPORATION,_L=REDMOND,_S=WASHINGTON,_C=US"
        ProductName="*" BinaryName="*">
      <BinaryVersionRange LowSection="*" HighSection="*" />
    </FilePublisherCondition>
  </Conditions>
  <Exceptions>
    <FilePathCondition Path="%OSDRIVE%\ProgramData\*" />
    <FilePathCondition Path="%PROGRAMFILES%\*" />

```

```

    <FilePathCondition Path="%WINDIR%\*" />
  </Exceptions>
</FilePublisherRule>
<FilePathRule Id="06dce67b-934c-454f-a263-2515c8796a5d" Name
  ="(Default_Rule)_All_scripts_located_in_the_Program_Files
  _folder" Description="Allows_members_of_the_Everyone_
  group_to_run_scripts_that_are_located_in_the_Program_
  Files_folder." UserOrGroupSid="S-1-1-0" Action="Allow">
  <Conditions>
    <FilePathCondition Path="%PROGRAMFILES%\*" />
  </Conditions>
</FilePathRule>
<FilePathRule Id="9428c672-5fc3-47f4-808a-a0011f36dd2c" Name
  ="(Default_Rule)_All_scripts_located_in_the_Windows_
  folder" Description="Allows_members_of_the_Everyone_group
  _to_run_scripts_that_are_located_in_the_Windows_folder."
  UserOrGroupSid="S-1-1-0" Action="Allow">
  <Conditions>
    <FilePathCondition Path="%WINDIR%\*" />
  </Conditions>
</FilePathRule>
<FilePathRule Id="ed97d0cb-15ff-430f-b82c-8d7832957725" Name
  ="(Default_Rule)_All_scripts" Description="Allows_members
  _of_the_local_Administrators_group_to_run_all_scripts."
  UserOrGroupSid="S-1-5-32-544" Action="Allow">
  <Conditions>
    <FilePathCondition Path="*" />
  </Conditions>
</FilePathRule>
</RuleCollection>
</AppLockerPolicy>

```

Appendix 5 - LOLBAS_AppLocker-policy.py

```
import csv
import argparse
import time
import uuid

def createAppLockerPolicy(csv_in):

    xml_publisherrule = """\
<FilePublisherRule Id="{uuid}" Name="Signed by O=MICROSOFT
CORPORATION, L=REDMOND, S=WASHINGTON, C=US" Description="
Block Microsoft signed file execution outside of allowed
locations (exceptions)" UserOrGroupSid="{sid}" Action="Deny">
  <Conditions>
    <FilePublisherCondition PublisherName="O=MICROSOFT
CORPORATION, L=REDMOND, S=WASHINGTON, C=US" ProductName
="*" BinaryName="*">
      <BinaryVersionRange LowSection="*" HighSection="*" />
    </FilePublisherCondition>
  </Conditions>
  <Exceptions>
    <FilePathCondition Path="%OSDRIVE%\ProgramData\*" />
    <FilePathCondition Path="%PROGRAMFILES%\*" />
    <FilePathCondition Path="%WINDIR%\*" />
  </Exceptions>
</FilePublisherRule>\n\
"""

    xml_template = """\
<AppLockerPolicy Version="1">
  <RuleCollection Type="Exe" EnforcementMode="NotConfigured">
    {XMLExecRules}
  </RuleCollection>
  <RuleCollection Type="Msi" EnforcementMode="NotConfigured">
    {XMLWinInstRules}
  </RuleCollection>
</AppLockerPolicy>
"""
```

```

<RuleCollection Type="Script" EnforcementMode="NotConfigured">
  {XMLScriptRules}
</RuleCollection>
<RuleCollection Type="Dll" EnforcementMode="NotConfigured">
  {XMLDLLsRules}
</RuleCollection>
<RuleCollection Type="Appx" EnforcementMode="NotConfigured" />
</AppLockerPolicy>
"""

```

```

xml_template_defaults = """\
<AppLockerPolicy Version="1">
  <RuleCollection Type="Exe" EnforcementMode="NotConfigured">
    {XMLExecRules}
    <FilePathRule Id="921cc481-6e17-4653-8f75-050b80acca20" Name
      ="(Default Rule) All files located in the Program Files
      folder" Description="Allows members of the Everyone group
      to run applications that are located in the Program
      Files folder." UserOrGroupSid="S-1-1-0" Action="Allow">
      <Conditions>
        <FilePathCondition Path="%PROGRAMFILES%\*" />
      </Conditions>
    </FilePathRule>
    <FilePathRule Id="a61c8b2c-a319-4cd0-9690-d2177cad7b51" Name
      ="(Default Rule) All files located in the Windows folder"
      Description="Allows members of the Everyone group to run
      applications that are located in the Windows folder."
      UserOrGroupSid="S-1-1-0" Action="Allow">
      <Conditions>
        <FilePathCondition Path="%WINDIR%\*" />
      </Conditions>
    </FilePathRule>
    <FilePathRule Id="fd686d83-a829-4351-8ff4-27c7de5755d2" Name
      ="(Default Rule) All files" Description="Allows members
      of the local Administrators group to run all applications
      ." UserOrGroupSid="S-1-5-32-544" Action="Allow">
      <Conditions>
        <FilePathCondition Path="*" />
      </Conditions>
    </FilePathRule>
  </RuleCollection>

```

```

<RuleCollection Type="Msi" EnforcementMode="NotConfigured">
  <FilePublisherRule Id="b7af7102-efde-4369-8a89-7a6a392d1473"
    Name="(Default Rule) All digitally signed Windows
    Installer files" Description="Allows members of the
    Everyone group to run digitally signed Windows Installer
    files." UserOrGroupSid="S-1-1-0" Action="Allow">
    <Conditions>
      <FilePublisherCondition PublisherName="*" ProductName
        = "*" BinaryName="*">
        <BinaryVersionRange LowSection="0.0.0.0" HighSection
          = "*" />
      </FilePublisherCondition>
    </Conditions>
  </FilePublisherRule>
  <FilePathRule Id="5b290184-345a-4453-b184-45305f6d9a54" Name
    = "(Default Rule) All Windows Installer files in %
    systemdrive%\Windows\Installer" Description="Allows
    members of the Everyone group to run all Windows
    Installer files located in %systemdrive%\Windows\
    Installer." UserOrGroupSid="S-1-1-0" Action="Allow">
    <Conditions>
      <FilePathCondition Path="%WINDIR%\Installer\*" />
    </Conditions>
  </FilePathRule>
  <FilePathRule Id="64ad46ff-0d71-4fa0-a30b-3f3d30c5433d" Name
    = "(Default Rule) All Windows Installer files" Description
    = "Allows members of the local Administrators group to run
    all Windows Installer files." UserOrGroupSid="S
    -1-5-32-544" Action="Allow">
    <Conditions>
      <FilePathCondition Path="*. *" />
    </Conditions>
  </FilePathRule>
  {XMLWinInstRules}
</RuleCollection>
<RuleCollection Type="Script" EnforcementMode="NotConfigured">
  {XMLScriptRules}
  <FilePathRule Id="06dce67b-934c-454f-a263-2515c8796a5d" Name
    = "(Default Rule) All scripts located in the Program Files
    folder" Description="Allows members of the Everyone
    group to run scripts that are located in the Program

```

```

    Files folder." UserOrGroupSid="S-1-1-0" Action="Allow">
<Conditions>
    <FilePathCondition Path="%PROGRAMFILES%\*" />
</Conditions>
</FilePathRule>
<FilePathRule Id="9428c672-5fc3-47f4-808a-a0011f36dd2c" Name
    ="(Default Rule) All scripts located in the Windows
    folder" Description="Allows members of the Everyone group
    to run scripts that are located in the Windows folder."
    UserOrGroupSid="S-1-1-0" Action="Allow">
<Conditions>
    <FilePathCondition Path="%WINDIR%\*" />
</Conditions>
</FilePathRule>
<FilePathRule Id="ed97d0cb-15ff-430f-b82c-8d7832957725" Name
    ="(Default Rule) All scripts" Description="Allows members
    of the local Administrators group to run all scripts."
    UserOrGroupSid="S-1-5-32-544" Action="Allow">
<Conditions>
    <FilePathCondition Path="*" />
</Conditions>
</FilePathRule>
</RuleCollection>
<RuleCollection Type="Dll" EnforcementMode="NotConfigured">
    {XMLDLLsRules}
<FilePathRule Id="bac4b0bf-6f1b-40e8-8627-8545fa89c8b6" Name
    ="(Default Rule) Microsoft Windows DLLs" Description="
    Allows members of the Everyone group to load DLLs located
    in the Windows folder." UserOrGroupSid="S-1-1-0" Action
    ="Allow">
<Conditions>
    <FilePathCondition Path="%WINDIR%\*" />
</Conditions>
</FilePathRule>
<FilePathRule Id="3737732c-99b7-41d4-9037-9cddfb0de0d0" Name
    ="(Default Rule) All DLLs located in the Program Files
    folder" Description="Allows members of the Everyone group
    to load DLLs that are located in the Program Files
    folder." UserOrGroupSid="S-1-1-0" Action="Allow">
<Conditions>
    <FilePathCondition Path="%PROGRAMFILES%\*" />

```



```

    </Conditions>
</FilePathRule>
<FilePathRule Id="fe64f59f-6fca-45e5-a731-0f6715327c38" Name
    ="(Default Rule) All DLLs" Description="Allows members of
    the local Administrators group to load all DLLs."
    UserOrGroupSid="S-1-5-32-544" Action="Allow">
    <Conditions>
        <FilePathCondition Path="*" />
    </Conditions>
</FilePathRule>
</RuleCollection>
<RuleCollection Type="Appx" EnforcementMode="NotConfigured">
    <FilePublisherRule Id="a9e18c21-ff8f-43cf-b9fc-db40eed693ba"
        Name="(Default Rule) All signed packaged apps"
        Description="Allows members of the Everyone group to run
        packaged apps that are signed." UserOrGroupSid="S-1-1-0"
        Action="Allow">
        <Conditions>
            <FilePublisherCondition PublisherName="*" ProductName
                ="*" BinaryName="*">
                <BinaryVersionRange LowSection="0.0.0.0" HighSection
                    ="*" />
            </FilePublisherCondition>
        </Conditions>
    </FilePublisherRule>
</RuleCollection>
</AppLockerPolicy>\
""

```

```

# String template for file path rules in AppLocker config
filePathRule = """\
<FilePathRule Id="{uuid}" Name="{filename}" Description="{
description}" UserOrGroupSid="{sid}" Action="Deny">
    <Conditions>
        <FilePathCondition Path="{filepath}" />
    </Conditions>
</FilePathRule>\n\
""

```

```

# create variables to store rules in the collections of

```

```

AppLocker
ExecCollection = ["exe", "com"]
ExecRules = ''

WinInstCollection = ["msi", "mst", "msp"]
WinInstRules = ''

ScriptsCollection = ["ps1", "bat", "cmd", "vbs", "js"]
ScriptsRules = ''

DLLsCollection = ["dll", "ocx"]
DLLsRules = ''

# Packaged apps not applicable, rules can only be done based on
# publisher, also no examples included in LOLBAS project

# Open input CSV
with open(csv_in) as input_csv:
    data = csv.reader(input_csv)

    if not (args.excludepublisher):
        ExecRules += xml_publisherrule.format(uuid=str(uuid.
            uuid4()), sid=args.sid)
        ScriptsRules += xml_publisherrule.format(uuid=str(
            uuid.uuid4()), sid=args.sid)
        DLLsRules += xml_publisherrule.format(uuid=str(uuid.
            uuid4()), sid=args.sid)

    # Loop through lines of CSV
    for row in data:
        if(row[4] == "AppLocker"):

            FileName = row[1]
            FilePath = row[3]

            # Find correct collection for AppLocker rule
            if((FilePath.split("."))[-1].lower() in
                ExecCollection):

                rule = filePathRule.format(uuid = str(uuid.
                    uuid4()), filename = FileName,

```

```

        description = "Rule_automatically_created
        _by_LOLBAS_AppLocker-policy.py_script",
        sid = args.sid, filepath = FilePath)
    ExecRules += rule

elif((FilePath.split("."))[-1].lower() in
    WinInstCollection):

    rule = filePathRule.format(uuid = str(uuid.
        uuid4()), filename = FileName,
        description = "Rule_automatically_created
        _by_LOLBAS_AppLocker-policy.py_script",
        sid = args.sid, filepath = FilePath)
    WinInstRules += rule

elif((FilePath.split("."))[-1].lower() in
    ScriptsCollection):

    rule = filePathRule.format(uuid = str(uuid.
        uuid4()), filename = FileName,
        description = "Rule_automatically_created
        _by_LOLBAS_AppLocker-policy.py_script",
        sid = args.sid, filepath = FilePath)
    ScriptsRules += rule

elif((FilePath.split("."))[-1].lower() in
    DLLsCollection):

    rule = filePathRule.format(uuid = str(uuid.
        uuid4()), filename = FileName,
        description = "Rule_automatically_created
        _by_LOLBAS_AppLocker-policy.py_script",
        sid = args.sid, filepath = FilePath)
    DLLsRules += rule

#if not (args.excludepublisher):
#     if(args.excludedefaults):
#         output = xml_template.format(XMLExecRules=ExecRules
, XMLWinInstRules=WinInstRules, XMLScriptRules=
ScriptsRules, XMLDLLsRules=DLLsRules, XMLPublisherRule=
xml_publisherrule.format(uuid=str(uuid.uuid4()), sid=args

```

```

        .sid))
#     else:
#         output = xml_template_defaults.format(XMLExecRules=
ExecRules, XMLWinInstRules=WinInstRules, XMLScriptRules=
ScriptsRules, XMLDLLsRules=DLLsRules, XMLPublisherRule=
xml_publisherrule.format(uuid=str(uuid.uuid4()), sid=args
.sid))
#else:
if(args.excludedefaults):
    output = xml_template.format(XMLExecRules=ExecRules,
XMLWinInstRules=WinInstRules, XMLScriptRules=
ScriptsRules, XMLDLLsRules=DLLsRules)
else:
    output = xml_template_defaults.format(XMLExecRules=
ExecRules, XMLWinInstRules=WinInstRules,
XMLScriptRules=ScriptsRules, XMLDLLsRules=DLLsRules)
return output

if __name__ == "__main__":
    # Get and parse arguments
    parser = argparse.ArgumentParser(description="Helper_script_
to_create_policy_XML_for_AppLocker_from_CSV_input,_script
_is_from:_https://github.com/hrunkaru/LOTLDefence",
formatter_class=argparse.
ArgumentDefaultsHelpFormatter
)
    parser.add_argument("-o", "--output", help="Output_XML_file_
path_for_AppLocker_policy_(default_is_current_folder)")
    parser.add_argument("-e", "--excludedefaults", help="Exclude
default_AppLocker_rules._Strongly_suggested_to_include_
default_rules,_unless_similar_rules_are_in_existing_
policy_you_plan_to_merge_this_one_with.", default=False,
action='store_true')
    parser.add_argument("-p", "--excludepublisher", help="
Exclude_Publisher_rule_to_block_Microsoft_signed_binaries
_from_non-native_locations.", default=False, action='
store_true')
    parser.add_argument("-s", "--src", help="Path_to_CSV_output_
from_LOLBAS-filepaths.py_script_(-p_switch),_where_

```

```

        required_rows_are_marked_with_AppLocker_or_WDAC_manually"
        , required=True)
parser.add_argument("-t", "--sid", help="Target_SID_group_
        for_the_created_AppLocker_policies.", required=True) #
        Add argument for user/group SID
args = parser.parse_args()

# Paths
## Input
CSV_in = args.src

## Timestring to use in names
timestr = time.strftime("%Y%m%d_%H%M%S")

## WDAC Output file
if(args.output):
    output_file = args.output
else:
    output_file = 'AppLocker_policy' + timestr + '.xml'

# Create data to output
output_data = createAppLockerPolicy(CSV_in)

# Save output to file
with open (output_file, 'w') as policy_out:
    policy_out.write(output_data)
print ("All_done,_find_the_output_file_at:_ " + output_file)

```