Windows Firewall Rules and MSIX Packages

A research paper by Timothy Mangan

February 15, 2026

Timothy Mangan

TMurgent Technologies, LLP

Introduction

The Windows Firewall controls traffic into and out of the operating system, using a set of OS wide, and Application specific rules. The firewall is controlled by a combination of (group) policy based rules and locally set policies, and local rules.

Often, vendor applications that use a specific IP port or ports, will add local application-specific rules as part of the installation of the application. These rules are added by the application installer software, using one of several APIs available to them. By adding the rule locally, they ensure that their application will not be blocked.

IT Organizations sometimes like to control the rules used by the firewall by setting them via group policy, but in reality are setting some via group policy and allowing the application installers to add to them out of convenience.

MSIX packages also may include firewall rules to be applied to the system, as part of the AppXMannifest file. This allows for the rule to correctly target the executable in the package, which may potentially be installed in different locations on different systems, due to the controls available to support multiple namespace Volumes for the packaged product installations.

For applications where the installer adds firewall rules using the available installer APIs, when recapturing the installation using the Microsoft MSIX Packaging Tool (MMPT), these rules end up being filtered out automatically and not brought along into the package.

Meanwhile, we see vendors releasing MSIX packages produced directly that include the firewall rules in their AppXManifest files.

In this paper, I look at new tooling options to help IT organizations deal with firewall rules in MSIX packages, whether from their own repackaging efforts or is intake of vendor supplied packages, whether they want these rules in the packages or want to remove them. Ultimately we see that the enterprise organization wants to be aware of vendor firewall rules, and then:

- Ensure they are part of the package.
- Ensure they are removed from the package so that they control them via central policy.

In this paper, you will find out how to do either of these.

About Firewall Rules

Application installers have several options to add the firewall rules.

1. **netsh advfirewall add rule**

   netsh advfirewall firewall add rule name="My App" dir=in action=allow program="C:\MyApp\app.exe" enable=yes

   Documentation link here.

2. **New-NetFirewallRule**

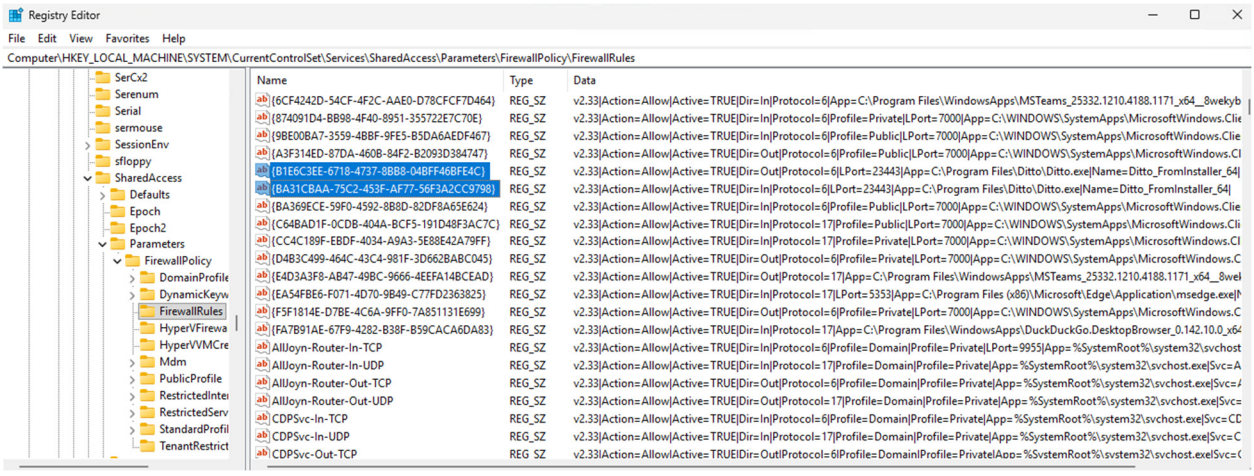
   New-NetFirewallRule -DisplayName "Allow Messenger" `
   -Direction Inbound `
   -Program "C:\Program Files (x86)\Messenger\msmsgs.exe" `
   -RemoteAddress LocalSubnet -Action Allow

   Documentation link here.

3. **Programmatically using a COM interfaced exposed by FirewallAPI.dll**

   In an MSI install package this would be part of a custom action.

No matter which API used, these rules are ultimately written to the Windows Registry as seen in this example from the Ditto installer:



There is also a SharedAccess\Defaults key that will contain rules that are not application-specific.

About MSIX AppXManifest and Firewall Rules

Microsoft extended the AppXManifest schemas to allow for the inclusion of firewall rules with the Desktop2 schema extension. This extension is implemented on all currently supported versions of the OS.

The schema implementation uses a package-level extension in the AppXManifest, where rules targeting the relative path to exe files within the package are targeted.

It starts with a package extension using Desktop2:Extension with a Category parameter of "windows.firewallRules". Under that extension element is a Desktop2:FirewallRules sub-element that has an Executable parameter that identifies the relative path to the exe file in the package. That sub-element will then have its own sub-elements of type Desktop2:Rule that have the individual rules.

Here is an example by the vendor package from DuckDuckGo published in the Winget repository:

```
<Extensions >
	<desktop2:Extension Category="windows.firewallRules">
		<desktop2:FirewallRules Executable="WindowsBrowser\WebView2\msedgewebview2.exe">
			<desktop2:Rule Direction="in" IPProtocol="TCP" Profile="all" />
			<desktop2:Rule Direction="in" IPProtocol="UDP" Profile="all" />
		</desktop2:FirewallRules>
	</desktop2:Extension>
	<desktop2:Extension Category="windows.firewallRules">
		<desktop2:FirewallRules Executable="WindowsBrowser\DuckDuckGo.WebView.exe">
			<desktop2:Rule Direction="in" IPProtocol="TCP" Profile="all" />
			<desktop2:Rule Direction="in" IPProtocol="UDP" Profile="all" />
		</desktop2:FirewallRules>
	</desktop2:Extension>
</Extensions>
```

Note that the windows firewall does not see any registry settings within the virtual registry of the package, if they exist. It is only the inclusion of the rules in the AppXManifest that matters. In fact, all of the vendor packages that I have seen to date that have firewall rules in the AppXManifest do not even have a virtual registry in the package.

About MSIX Packaging Tool Exclusions

The Microsoft MSIX Packaging Tool generally captures changes made to the file system and registry during a monitoring session to include in the package. The tool has configuration settings that contains a file and a registry list of exclusions.

Ultimately, firewall rules are written under the registry key **HKLM\System\Services\CurrentControlSet\SharedAccess** registry key. When using the installer APIs, including system dll, newsh, and powerershell APIs, these entries are written by a generic svchost process.

There is also a poorly understood Service Exclusion list added in 2003 that attempts to provide control over changes made by known windows services that had always been excluded by the tool in the past. When the application installer uses one of the APIs to add the firewall rules, a Process Monitor trace will show that the rules are written into the Windows Registry by a generic svchost process.

Removing the firewall service from the MMPT configuration (named MpsSvc) from the MMPT service exclusion list does not cause capture of these keys.

However, directly writing those keys does allow for capture, even if not the right way to set the rules.

Using PassiveInstall to ensure vendor rules are not missed

Version 3.0 of PassiveInstall provides some PowerShell cmdlets that may be used to overcome this exclusion by the MMPT.

PassiveInstall is a free opens source tool I originally wrote in 2017. The tool is quite useful when creating wrapper scripts to customize the vender installer for a variety of things that we want to do during the repackaging operation.

This new release adds two new PowerShell cmdlets that can be added to these wrapper scripts to allow for the MMPT capture of the vendor firewall rules without you having to know what the rules a particular application adds, or even if it adds any at all.

The first new cmdlet, **Get-PassiveFirewallRules**, returns a snapshot of the registry rules. You add this cmdlet twice in your wrapper script, once before the customized install and once after.

The second new cmdlet, **Redo-PassiveNewFirewallRules** is then added after that. It uses the two snapshots to determine any new rules that were added, and will rewrite the registry values in a way that the MMPT will capture those rules into the virtual registry.

Here is the example from Get-Help on the cmdlets:

```
-------------- EXAMPLE 1 --------------

This is a typical usage scenario:PS>
PS> $BeforeSnapshot = Get-PassiveFirewallRules
PS> msiexec /q vendorInstaller.msi
PS> $AfterSnapshot = Get-PassiveFirewallRules
PS> $Changes = Redo-PassiveNewFirewallRules $BeforeSnapshot $AfterSnapshot
PS> $Changes.Show()
PS>
```

Unfortunately, the MMPT will still not write those rules into the AppXManifest file, but we will address this in the TMEditX Editor post-processing.

Using TMEditX Editor for Post-processing

Repackaging into MSIX usually involves post-processing the captured package in order to obtain complete compatibility. Because we previously did not see the MMPT capture the firewall rules, we never needed a fixup for adding the rules into the AppXManifest.

TMEditX Editor is my licensed software to fix up those packages, filled with a package analyzer that offers automatic and manual fixing to your packages, along with a full editor. Version 7.1 adds in a new option to control package firewall rules automatically.

Tool Configuration

The option is controlled in the options tab of the tool:

This option provides the ability to fix firewall rules whichever way you want:

- Opt-in if you want rules detected in the virtual registry to be added to the AppXManifest file so that they are effective.
- Out-out if you want rules detected in the AppXManifest file to be removed. Remember that vendor provided packages might have these rules.

In addition to the configuration option, the tool has a pair of command line options to override the configuration setting, /AutoAddFirewallRules and /AutoRemoveFirewallRules. These may be useful in automated package fixing scripts.

Analysis

The package analysis support serves to let you know that the package has firewall rules and what they are. If you want to control these via policy, you need to be aware that the installer added them.

Opt-in Analyis

For the Opt-in setting, if the package have the virtual registry settings but they are not in the AppXManifest, they appear in a Recommended fixup like this:

Opt-out Analysis

For the Opt-out setting, if the package has entries in the AppXManifest, they appear in a Recommended fixup like this:

Summary

IT Desktop Engineering can now have information about, and control over application firewall rules added by both traditional application installers and vendor pre-packaged MSIX packages.

About

Tim Mangan leads the technical side at TMurgent Technologies, LLP. TMurgent creates research oriented papers such as this for the technical communities to better understand Microsoft technologies. Whether to repackage applications, or build your own tooling around Microsoft, check out our white papers at TMurgent.com