PowerShell Constrained Language Mode

By PowerShell Team

Published: 2017-11-02 · Archived: 2026-04-06 03:25:14 UTC

PowerShell Constrained Language Mode

Update (May 17, 2018)

In addition to the constraints listed in this article, system wide Constrained Language mode now also disables the ScheduledJob module. The ScheduledJob feature uses Dot Net serialization that is vulnerable to deserialization attacks. So now whenever an application whitelisting solution is applied such as DeviceGuard or AppLocker, PowerShell will run in Constrained Language mode and also disable the ScheduledJob module. Use the Windows Task Scheduler or PowerShell ScheduledTasks module as an alternative. For more information see CVE-2018-0958.

What is PowerShell Constrained Language?

PowerShell Constrained Language is a language mode of PowerShell designed to support day-to-day administrative tasks, yet restrict access to sensitive language elements that can be used to invoke arbitrary Windows APIs.

You can place a PowerShell session into Constrained Language mode simply by setting a property:

```
PS C:\> $ExecutionContext.SessionState.LanguageMode
FullLanguage
PS C:\> $ExecutionContext.SessionState.LanguageMode = "ConstrainedLanguage"
PS C:\> $ExecutionContext.SessionState.LanguageMode
ConstrainedLanguage

PS C:\> [System.Console]::WriteLine("Hello")
Cannot invoke method. Method invocation is supported only on core types in this language mode.
At line:1 char:1
+ [System.Console]::WriteLine("Hello")
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : InvalidOperation: (:) [], RuntimeException
    + FullyQualifiedErrorId : MethodInvocationNotSupportedInConstrainedLanguage
```

Of course, this is not secure. A user can simply start another PowerShell session which will run in Full Language mode and have full access to PowerShell features. As part of the implementation of Constrained Language, PowerShell included an environment variable for debugging and unit testing called `__PSLockdownPolicy`. While we have never documented this, some have discovered it and described this as an enforcement mechanism. This is

unwise because an attacker can easily change the environment variable to remove this enforcement. In addition, there are also file naming conventions that enable FullLanguage mode on a script, effectively bypassing Constrained Language. Again, this is for unit testing. These test hooks cannot override a Device Guard UMCI policy and can only be used when no policy enforcement is applied.

Then what is it for?

PowerShell Constrained Language mode was designed to work with system-wide application control solutions such as Device Guard User Mode Code Integrity (UMCI). Application control solutions are an incredibly effective way to drastically reduce the risk of viruses, ransomware, and unapproved software. For DeviceGuard UMCI the approved applications are determined via a UMCI policy. PowerShell automatically detects when a UMCI policy is being enforced on a system and will run only in Constrained Language mode. So PowerShell Constrained Language mode becomes more interesting when working in conjunction with system-wide lockdown policies.

PowerShell's detection of system policy enforcement through DeviceGuard is supported only for Windows platform running Windows PowerShell version 5.1 or PowerShell 7. It does not work on non-Windows platforms. So this is currently very much a Windows security feature. However, we will continue to enhance this for non-Windows platforms where feasible.

These lockdown policies are important for high-value systems that need to be protected against malicious administrators or compromised administrator credentials. With a policy enforced even administrators are limited to what they can do on the system.

Since Constrained Language is so limited, you will find that many of the approved scripts that you use for advanced systems management no longer work. The solution to this is simple: add these scripts (or more effectively: your code signing authority that signed them) to your Device Guard policy. This will allow your approved scripts to run in Full Language mode.

For example, all PowerShell module files shipped with Windows (e.g., Install-WindowsFeature) are trusted and signed. The UMCI policy allowing signed Windows files lets PowerShell run these modules in Full Language mode. But if you create a custom PowerShell module that is not allowed by the policy then it will be considered untrusted and run with Constrained Language restrictions.

Consequently, any PowerShell module marked as trusted in the policy needs to be carefully reviewed for security vulnerabilities. A vulnerability could allow code injection, or leak private functions not intended for public use. In either case it could allow a user to run arbitrary code in Full Language mode, thus bypassing the system policy protections.

We have described these dangers in much more detail in our post, "Writing Secure PowerShell Scripts" (coming soon).

What does Constrained Language constrain?

Constrained Language consists of a number of restrictions that limit unconstrained code execution on a locked-down system. These restrictions are:

- PowerShell module script files must explicitly export functions by name without the use of wildcard characters. This is to prevent inadvertently exposing powerful helper function not meant to be used publicly.
- PowerShell module manifest files must explicitly export functions by name without the use of wildcards. Again, to prevent inadvertent exposure of functions.
- COM objects are blocked. They can expose Win32 APIs that have likely never been rigorously hardened as part of an attack surface.
- Only approved .NET types are allowed. Many .NET types can be used to invoke arbitrary Win32 APIs. As a result only specific whitelisted types are allowed.
- Add-Type is blocked. It allows the creation of arbitrary types defined in different languages.
- The use of PowerShell classes are disallowed. PowerShell classes are just arbitrary C# type definitions.
- PowerShell type conversion is not allowed. Type conversion implicitly creates types and runs type constructors.
- Dot sourcing across language modes is disallowed. Dot sourcing a script file brings all functions, variables, aliases from that script into the current scope. So this blocks a trusted script from being dot sourced into an untrusted script and exposing all of its internal functions. Similarly, an untrusted script is prevented from being dot sourced into a trusted script so that it cannot pollute the trusted scope.
- Command resolution automatically hides commands you cannot run. For example, a function created in Constrained Language mode is not visible to script running in Full Language mode.
- XAML based workflows are blocked since they cannot be constrained by PowerShell. But script based workflows and trusted XAML based workflows shipped in-box are allowed.
- The SupportedCommand parameter for Import-LocalizedData is disabled. It allows additional commands prevented by Constrained Language.
- Invoke-Expression cmdlet always runs in Constrained Language. Invoke-Expression cannot validate input as trusted.
- Set-PSBreakpoint command is blocked unless there is a system-wide lockdown through UMCI.
- Command completers are always run in Constrained Language. Command completers are not validated as trustworthy.
- Commands and script run within the script debugger will always be run in Constrained Language if there is a system-wide lockdown.
- The DSC Configuration keyword is disabled.
- Supported commands and Statements are not allowed in script DATA sections.
- Start-Job is unavailable if the system is not locked-down. Start-Job starts PowerShell in a new process and if the system is not locked-down the new process runs in Full Language mode.

As we can see, Constrained Language mode imposes some significant restrictions on PowerShell. Nevertheless, it remains a formidable and capable shell and scripting language. You can run native commands and PowerShell cmdlets and you have access to the full scripting features: variables, statements, loops, functions, arrays, hashtables, error handling, etc.

How is this different from JEA?

PowerShell Constrained Language restricts only some elements of the PowerShell language along with access to Win32 APIs. It provides full shell access to all native commands and many cmdlets. It is not designed to operate independently and needs to work with application control solutions such as UMCI to fully lockdown a system and prevent access to unauthorized applications. Its purpose is to provide PowerShell on a locked-down system without compromising the system.

JEA (Just Enough Administration) is a sandboxed PowerShell remote session that is designed to strictly limit what the logged on user can do. It is configured in 'no language mode', has no access to file or other drive providers, and makes only a small set of cmdlets available. These cmdlets are often custom and designed to perform specific management functions without giving unfettered access to the system. The set of cmdlets provided in the session is role based (RBAC) and the session can be run in virtual or Group Managed Service accounts.

The JEA scenario is where an administrator needs to perform a management task on a high-value machine (such as collect logs or restart a specific service). The administrator creates a remote PowerShell session to the machine's JEA endpoint. Within that session the user has access to only those commands needed to perform the task and cannot directly access the file system or the registry or run arbitrary code.

Summary

PowerShell Constrained Language is designed to work with application whitelisting solutions in order to restrict what can be accessed in an interactive PowerShell session with policy enforcement. You configure which scripts are allowed full system access through the whitelisting solution policy. In contrast, JEA is a sandboxed PowerShell remote session that restricts an interactive session to specific commands based on user role.

Paul Higinbotham [MSFT] PowerShell Team

Category

Author

PowerShell Team

PowerShell is a task-based command-line shell and scripting language built on .NET. PowerShell helps system administrators and power-users rapidly automate tasks that manage operating systems (Linux, macOS, and Windows) and processes.

Source: https://devblogs.microsoft.com/powershell/powershell-constrained-language-mode/