Microsoft Edge Cleanup and Reinstall Script

This page documents a robust PowerShell script used to **completely remove an application** (like Microsoft Edge) and reinstall it cleanly. It's especially useful when traditional uninstallation or update processes fail, leaving broken traces behind.

? About

This script helps remove:

- Registry entries
- Scheduled tasks
- Background services
- Running processes
- Residual folders and files

? Configuration Tips

Finding Configurations:

1. Search online for:
   - Uninstall GUIDs
   - File paths
   - Registry keys
2. Define:
   - `$regNameToMatch`
   - `$knownUninstallers`
   - `$knownPathsToDelete`, etc.
3. Repeat if needed: leftover services or background tasks can require multiple runs.

```
Get-ItemProperty "HKLM:\SOFTWARE\WOW6432Node\Microsoft\Windows\CurrentVersion\Uninstall\*" -
ErrorAction SilentlyContinue |
    Where-Object { $_.DisplayName -eq "Microsoft Edge" } |
    Select-Object PSChildName, DisplayName, DisplayVersion, SystemComponent, UninstallString,
WindowsInstaller |
    Format-List
```

```
powershell.exe -NoProfile -EncodedCommand
JABrACAAPQAgAEcAZQB0AC0ASQB0AGUAbQBQAHIAbwBwAGUAcgB0AHkAIAAnAEgASwBMAE0AOgBcAFMATwBGAFQAVwBBAF
IARQBcAFcATwBXADYANAAzADIATgBvAGQAZQBcAE0AaQBjAHIAbwBzAG8AZgB0AFwAVwBpAG4AZABvAHcAcwBcAEMAdQBy
AHIAZQBuAHQAVgBlAHIAcwBpAG8AbgBcAFUAbgBpAG4AcwB0AGEAbABsAFwATQBpAGMAcgBvAHMAbwBmAHQAIABFAGQAZw
BlACcAIAAtAEUAcgByAG8AcgBBAGMAdABpAG8AbgAgAFMAaQBsAGUAbgB0AGwAeQBDAG8AbgB0AGkAbgB1AGUACgBpAGYA
IAAoACQAawApACAAewAKACAAIAAgACAAJABlAHgAZQBQAGEAdABoACAAPQAgACQAawAuAFUAbgBpAG4AcwB0AGEAbABsAF
MAdABpAGkAbgBnACAALQByAGUAcABsAGEAYwBlACAAJwBeACIAKABbAF4AIgBdACsAKQAiAC4AKgAnACwAJwAkADEAJwAK
ACAAIAAgACAAIgBTAHkAcwB0AGUAbQBDAG8AbQBwAG8AbgBlAG4AdAA9ACQAKAAkAGsALgBTAHkAcwB0AGUAbQBDAG8AbQ
BwAG8AbgBlAG4AdAApACAAfAAgAEUAeABlAEUAeABpAHMAdABzAD0AJAAoAFQAZQBzAHQALQBQAGEAdABoACAAJABlAHgA
ZQBQAGEAdABoACkAIAB8ACAARQB4AGUAUABhAHQAaAA9ACQAZQB4AGUAUABhAHQAaAAiAAoAfQAgAGUAbABzAGUAIAB7AA
oAIAAgACAAIAAnAEsAZQB5ACAAbgBvAHQAIABwAHIAZQBzAGUAbgB0ACcACgB9AA==
```

Remove duplicate key

```
powershell.exe -NoProfile -EncodedCommand
JABwACAAPQAgACcASABLAEwATQA6AFwAUwBPAEYAVABXAEEAUgBFAFwAVwBPAFcANgA0ADMAMgBOAG8AZABlAFwATQBpAG
MAcgBvAHMAbwBmAHQAXABXAGkAbgBkAG8AdwBzAFwAQwB1AHIAcgBlAG4AdABWAGUAcgBzAGkAbwBuAFwAVQBuAGkAbgBz
AHQAYQBsAGwAXABNAGkAYwByAG8AcwBvAGYAdAAgAEUAZABnAGUAJwAKAGkAZgAgACgAVABlAHMAdAAtAFAAYQB0AGgAIA
AkAHAAKQAgAHsACgAgACAAIAAgAFIAZQBtAG8AdgBlAC0ASQB0AGUAbQAgAC0AUABhAHQAaAAgACQAcAAgAC0ARgBvAHIA
YwBlAAoAIAAgACAAIAAiAFIAZQBtAG8AdgBlAGQAIABkAHUAcABsAGkAYwBhAHQAZQAgAGsAZQB5ADoAIAAkAHAAIgAKAH
0AIABlAGwAcwBlACAAewAKACAAIAAgACAAIgBLAGUAeQAgAG4AbwB0ACAAcAByAGUAcwBlAG4AdAAgAC0AIABuAG8AdABo
AGkAbgBnACAAdABvACAAcgBlAG0AbwB2AGUAIgAKAH0A
```

```
Remove-Item "HKLM:\SOFTWARE\WOW6432Node\Microsoft\Windows\CurrentVersion\Uninstall\Microsoft
Edge" -Force
```

?? Script

```
###
### About
###
#    Application Cleanup and Reinstall
#   What is this?
```

```
#       This PowerShell script helps remove most traces of an installed application
#       so you can reinstall it cleanly. It targets leftover files, folders, and registry
#       entries that can block updates or fresh installs. It was originally built to
#       remove problematic versions of Microsoft Edge that failed to update.
#   How to find configurations?
#       1. Search online for uninstall GUIDs, file paths, and registry keys related to
#       your application. Official docs or forums often list these details.
#       2. Use RegShot (on GitHub) to capture "before" and "after" system snapshots.
#       Compare them to see which files and registry entries the installer added.
#       3. In the script, specify top-level folders, file patterns, and registry keys you
#       want removed. It will also scan the usual uninstall locations by name.
#       4. If the first run doesn't catch everything, repeat the process. Look for
#       background processes, services, or custom registry keys not covered initially.
#   Having issues?
#       Please email me with detailed information:
#       • RegShot comparison report
#       • Script output or action log
#       • List of running processes
#       • Your script configurations
#       The more data you provide, the faster we can diagnose and fix the problem.
##############################################################################################
########################
##############################################################################################
########################
###
### Configurations
###
##############################################################################################
########################
$regNameToMatch = "Microsoft Edge"
#   This setting tells the script what application name to look for when scanning standard
uninstall registry
#   locations. It uses a "match" operation against the Name, DisplayName, and ProductName
properties in those
#   registry keys. Use a clear, unique identifier for your app so you don't accidentally
remove other software.
#   Example usage
#       $regNameToMatch = "Microsoft Edge"
#   This will match any registry entries whose Name, DisplayName, or ProductName contains
"Microsoft Edge"
```

```
##########################################################################################
########################
$knownUninstallers = @('"C:\Program Files
(x86)\Microsoft\Edge\Application\*\Installer\setup.exe" --uninstall --system-level --force-
uninstall')
#   Define known uninstaller paths Add any known uninstaller executable locations to this
array. You can use PowerShell
#   wildcards (\*) to match varying folder names. If the path includes spaces, wrap it in
single quotes on the outside
#   and double quotes for the executable path.
#   Example usage
#       $knownUninstallers = @(
#           '"C:\Program Files (x86)\Microsoft\Edge\Application\*\Installer\setup.exe" --
uninstall --system-level --force-uninstall'
#       )
##########################################################################################
########################
$scheduledTasksNameToRemove = "Edge"
#   Configure scheduled task removalThis array specifies a name pattern to search for in
Scheduled Tasks.The script
#   uses a "like" operator, adding wildcards (*) before and afterthe name you provide to match
any part of the task name.
#   Example usage
#       $scheduledTasksNameToRemove = "Edge"
#   This will match tasks containing "Edge" anywhere in their names.
##########################################################################################
########################
$processNamesToStop = @("msedge","msedgewebview2","widget","msiexec","MicrosoftEdgeUpdate")
#   Define processes to stop List any running process names that should be terminated at the
start of cleanup.
#   You can include wildcards (*) for partial matches.
#   Example usage
#      $processNamesToStop =
@("msedge","msedgewebview2","widget","msiexec","MicrosoftEdgeUpdate")
#   This will stop any processes matching those names before uninstall steps begin.
##########################################################################################
########################
$knownPathsToDelete = @(
    "HKLM:\SOFTWARE\Microsoft\EdgeUpdate",
    "HKLM:\SOFTWARE\WOW6432Node\Microsoft\EdgeUpdate",
```

```
    "HKLM:\SOFTWARE\Microsoft\Edge",
    "HKLM:\SOFTWARE\WOW6432Node\Microsoft\Edge",
    "C:\Program Files (x86)\Microsoft\Edge",
    "C:\ProgramData\Microsoft\EdgeUpdate"
)
#   Define known paths to delete. Specify any registry keys or file paths to remove during
cleanup.
#   Use PowerShell paths for registry (e.g., HKLM:...) and standard file paths. Wildcards (*)
can be used.
#   Example usage
#       $knownPathsToDelete = @(
#          "HKLM:\SOFTWARE\Microsoft\EdgeUpdate",
#          "HKLM:\SOFTWARE\WOW6432Node\Microsoft\EdgeUpdate",
#          "C:\Program Files (x86)\Microsoft\Edge"
#       )
##########################################################################################
########################
$knownsServicesToDelete = @()
#   Define services to delete List any service names to remove during cleanup. The script uses
sc.exe to delete
#   matching services. Wildcards (*) are supported.
#   Example usage
#      $knownServicesToDelete = @("*Soup*")
#   This will delete any service whose name contains "Soup"
##########################################################################################
########################
$installerStrings = @("msiexec.exe /i MicrosoftEdgeEnterpriseX64.msi /qn")
#   Define install commands for reinstallation If you plan to reinstall the application, list
the full install
#   commands here. Include the executable and all arguments. Installer files must reside in
the script’s working
#   directory, so you can call them by name without a full path.
#   Example usage
#      $installerStrings = @("msiexec.exe /i MicrosoftEdgeEnterpriseX64.msi /qn")
##########################################################################################
########################
$executableTimeoutSeconds = 300
#   Configure executable timeout This setting defines the maximum time (in seconds) the script
will wait for any
#   executable to finish. Keep this value as low as possible to avoid hanging on installers
```

```
that launch interactive GUIs.
#   Example usage
#      $executableTimeoutSeconds = 300
#   This sets a 5-minute timeout for each executable call.
################################################################################################
########################
################################################################################################
########################
###
### Main function can be modified to change the order of operation or additional rounds
###
################################################################################################
########################
function Main {
    SetLog("Starting Cleanup")
    Stop-AppProcessesByName($processNamesToStop)
    Remove-ScheduledActions($scheduledTasksNameToRemove)
    Remove-KnownServices($knownsServicesToDelete)
    Invoke-KnownUninstallStrings($knownUninstallers)
    $cleanupPaths = Find-AllRegistryLocations($regNameToMatch)
    $GUIDs =  Get-GUIDFromRegPath($cleanupPaths)
    $cleanupPaths = $cleanupPaths +  (Expand-GUIDPaths($GUIDs))
    Invoke-MSIUninstall($GUIDs)
    Invoke-UninstallStrings($cleanupPaths)
    Remove-CleanupItems($knownPathsToDelete)
    Remove-CleanupItems($cleanupPaths)
    Invoke-InstallStrings($installerStrings)
    SetLog("Cleanup complete")
}
################################################################################################
########################
################################################################################################
########################
###
### Working function below this point. If you find modifications are needed, please
### reach out for assistance or share your changes. This will allow others to avoid
### and benefit from the issues you  encountered. jeff@chuco.com
###
################################################################################################
########################
```

```
function SetLog($logline){
    Write-Host ("[{0:MM/dd/yy} {0:HH:mm:ss}]" -f (Get-Date) + $logline)
}

function Remove-KnownServices($services){
    if($services.count -eq 0){
        return
    }
    SetLog("Removing Services")
    foreach($service in $services){
        $svcs = Get-Service $service
        SetLog("Found $($svcs.count) service(s) using $service")
        #looping through if multiple services are found
        foreach($svc in $svcs){
            $thisName = $svc.Name
            if($svc.status -eq "Running"){
                SetLog("Service $thisName is running. Attempting to stop before removal.")
                $svc | Stop-Service -ErrorAction SilentlyContinue
            }
            SetLog("Attempting to remove: $thisName")
            sc.exe delete ($thisName)
            if((Get-Service $thisName).count -eq 0){
                SetLog("Service removed")
            } else {
                SetLog("Failed to remove service")
            }
        }
    }
}

function Invoke-KnownUninstallStrings($strings){
    Invoke-ExecutableStrArr -arr $strings -msg "Starting known uninstall strings"
}
function Invoke-ExecutableStrArr{
    param(
        [array]$arr,
        [string]$msg
    )
    if($arr.count -eq 0){
        return
```

```
    }
    SetLog($msg)
    foreach($string in $arr){
        $stringParts = Expand-ExecutableString($string)
        Invoke-Executable @($stringParts.path,$stringParts.args)
    }
}

function Invoke-InstallStrings($installers){
    Invoke-ExecutableStrArr -arr $installers -msg "Starting installer(s)"
}

function Stop-AppProcessesByName($procNames){
    SetLog("Stopping Application Processes")
    $procNames = ConvertTo-Array $procNames
    foreach($procName in $procNames){
        SetLog("Looking for processes with name $procName")
        $attempt = 0
        do{
            if($attempt -gt 0 ){
                SetLog("Attempt number $($attempt+1) to stop processes for $procName")
            }

            $procs = Get-Process | Where-Object {$_.Name -match $procName}
            foreach($proc in $procs){
                try{
                    SetLog("Stopping process $($proc.processname) PID $($proc.id)")
                    Stop-Process -Id $proc.Id -Force -ErrorAction SilentlyContinue
                }catch{
                    SetLog("ERROR stopping process: $($Error[0].Exception.Message)")
                    SetLog("ERROR NAME: $($error[0].Exception.GetType().FullName)")
                }
            }
            Start-Sleep 1
            $attempt++
        }while(Get-Process | Where-Object {$_.Name -match $procName})
    }
}

function Invoke-MSIUninstall($GUIDs){
```

```
    if($GUIDs.count -eq 0){
        return
    }
    SetLog("Uninstall by GUID")
    foreach($GUID in $GUIDs){
        Invoke-Executable @("msiexec.exe","/x $GUID /q")
    }
}

function Invoke-UninstallStrings($cleanupObj){
    foreach($path in $cleanupObj){
        $targetStrings = New-Object System.Collections.ArrayList
        foreach($props in (Get-ItemProperty $path)){
            $propKeys = $props.PSObject.Properties.Name
            #Locate any well know uninstall string locations
            if($propKeys -contains "UninstallString"){
                $targetStrings.add($props.UninstallString) | Out-Null
            }
            if($propKeys -contains "QuietUninstallString"){
                $targetStrings.add($props.QuietUninstallString) | Out-Null
            }
            if($targetStrings.count -eq 0){
                SetLog("Unhandled potential props: $($propKeys -join ", ")")
            }
        }

        foreach($targetString in $targetStrings){
            if($targetString -match "MsiExec"){
                if($targetString -match ".*({.*}).*"){
                    Invoke-MSIUninstall @($Matches[1])
                }else{
                    Invoke-Executable @("msiexec.exe","/X $($targetString) /q")
                }
            } else {
                #Regex to extract the executable and arguments
                $exeStrings = Expand-ExecutableString($targetString)
                if($exeStrings){
                    Invoke-Executable @($exeStrings['Path'],$exeStrings['Args'])
                } else {
                    SetLog("Unable to process uninstall string: $($targetString)")
```

```
                }
            }
        }
    }
}

function Expand-ExecutableString($str){
    $pattern = '(?:"(?<Path>[A-Za-z]:\\[^"\\]+(?:\\[^"\\]+)*)"|(?<Path>[A-Za-
z]:\\\S+(?:\\\S+)*)|(?<Path>[\w\.-]+\.exe))(?:\s+(?<Args>.*))?'
    if($str -match $pattern){
      return [PSCustomObject]@{
            Path = $Matches['Path']
            Args = $Matches['Args']
        }
    }
    return $false
}

function Expand-GUIDPaths($GUIDs){
    $results = New-Object System.Collections.ArrayList
    foreach($GUID in $GUIDs){
        $productCode = Convert-GUIDToMsiProductCode($GUID)
        $productCodeRegistryPath = (Get-Item
"HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\Installer\UserData\S-1-5-
18\Products\$productCode").PSPath
        $results.add(($productCodeRegistryPath))|Out-Null
        $results + (Find-AllRegistryLocations($GUID)) + (Find-
AllRegistryLocations($productCode))
    }
    return $results
}

function Get-GUIDFromRegPath($pathsObj){
    $results = New-Object System.Collections.ArrayList
    foreach($path in $pathsObj){
        if($path -match ".*({.*}).*"){
            $results.add($Matches[1])|Out-Null
        }
    }
    if($results.Count -gt 0){
```

```
        return $results
    } else {
        return $null
    }
}

Function Find-AllRegistryLocations($appName){
    SetLog("Locating all application registry values for $appName")
    $results = New-Object System.Collections.ArrayList
    $paths = @(
            "HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall",
            "HKLM:\SOFTWARE\WOW6432Node\Microsoft\Windows\CurrentVersion\Uninstall",
            "HKLM:\SOFTWARE\Classes\Installer\Products",
            "HKLM:\SOFTWARE\Classes\Installer\UpgradeCodes",
            "HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\Installer\Folders"
        )
    #To work with the asterisk path needed for HKU an additional loop is required
    foreach($path in
@("Registry::HKEY_USERS\*\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall","Registry::HKEY
_USERS\*\SOFTWARE\Wow6432Node\Microsoft\Windows\CurrentVersion\Uninstall")){
        Get-ChildItem $path | ForEach-Object {$paths += $_.PSPath}
    }
    $valuesOfInterest = @("Name","DisplayName","ProductName")
     foreach ($path in $paths){
        if (Test-Path $path){
            $regHive = Get-ChildItem -Path $path
            foreach($regKey in $regHive){
                $thisPSPath = $regKey.PSPath
                if($thisPSPath -match $appName){
                    SetLog("FOUND: $thisPSPath")
                    $results.Add($thisPSPath)|Out-Null
                    continue
                }
                foreach($value in $valuesOfInterest){
                    try{
                        $foundValue = Get-ItemPropertyValue -Path $thisPSPath -Name $value
                        if($foundValue -match $appName){
                            $results.Add($thisPSPath)|Out-Null
                        }
                    }catch [System.Management.Automation.PSArgumentException]{
```

```
                        #Nothing to do - This reg did not have one of the values
                    }catch{
                        #Log it for later review
                        SetLog("ERROR: $($Error[0].Exception.Message)")
                        SetLog("ERROR NAME: $($error[0].Exception.GetType().FullName)")
                    }

                }
            }
        }
    }
    return $results | Select-Object -Unique
}

function Convert-GUIDToMsiProductCode {
    param([string]$Guid)
    $ProductIDChars = [regex]::replace($GUID, "[^a-zA-Z0-9]", "")
    $RearrangedCharIndex =
7,6,5,4,3,2,1,0,11,10,9,8,15,14,13,12,17,16,19,18,21,20,23,22,25,24,27,26,29,28,31,30
    Return -join ($RearrangedCharIndex | ForEach-Object{$ProductIDChars[$_]})
}

function ConvertTo-Array($in){
    if(-not($in -is [array])){
        return @($in)
    }
    return $in
}

function Invoke-Executable($un){
    SetLog("Expanding: $un")
    try{
        $thisPaths = Get-Item $un[0] -ErrorAction SilentlyContinue
    }catch{
        #nothing to do. catch to silence errors
    }
    if($thisPaths.count -eq 0 -and -not($un[0] -match "\*")){
        $thisPaths = @{}
        $thisPaths.FullName = $un[0]
    }
```

```
    if($thisPaths.count -eq 0){
        SetLog("No executable found")
        return
    }
    #Looping through to handle variable paths returning multiple
    foreach($thisPath in $thisPaths){
        $exePath = $thisPath.fullname
        if($null -eq $exePath -or $exePath -eq ""){
            continue
        }
        SetLog("Starting command: $exePath $($un[1])")
        if($un[1]){
            $thisPoc = Start-Process $exePath -ArgumentList ($un[1]) -PassThru -ErrorAction
SilentlyContinue
        } else {
            $thisPoc = Start-Process $exePath -PassThru -ErrorAction SilentlyContinue
        }
        if ($thisPoc){
            try{
                Wait-Process -Id $thisPoc.Id -Timeout $executableTimeoutSeconds
            }catch{
                SetLog("ERROR: executable exceeded timed out (max:
$executableTimeoutSeconds)")
                Stop-Process -Id $thisPoc.Id -Force
            }
        } else {
            SetLog("Failed to start executable process")
        }
        SetLog("ExitCode: $($thisPoc.ExitCode)")
    }
}

function Remove-ScheduledActions($name){
    SetLog("Checking for scheduled tasks with task name like: $name")
    $tasks = Get-ScheduledTask | Where-Object { $_.TaskName -like "*$name*" }
    if ($tasks) {
        foreach ($task in $tasks) {
            SetLog("Removing scheduled task: $($task.TaskName)")
            Unregister-ScheduledTask -TaskName $task.TaskName -Confirm:$false -ErrorAction
SilentlyContinue
```

```
        }
    } else {
        SetLog("No scheduled tasks found with task name like: $name")
    }
}

function Remove-CleanupItems($paths){
    if($paths.count -eq 0){
        return
    }
    SetLog("Removing items by path")
    $paths = ConvertTo-Array $paths
    foreach($path in $paths){
        if(Test-Path $path){
            SetLog("Found and removing item: $path")
            Remove-Item -Path $path -Recurse -Force -Erroraction SilentlyContinue
            SetLog("Item removed: $(-not(Test-Path $path))")
        } else {
            SetLog("Item found during scan no longer exists: $path")
        }
    }
}

Main
```

```
<#
.SYNOPSIS
    Converts a self-updating (setup.exe-based) Microsoft Edge system-level install
    to the MSI Enterprise build, so Tanium has authoritative control over the version.

.DESCRIPTION
    Targets endpoints where Edge was installed via the Chromium standalone/self-update
    installer (registers under HKLM Uninstall as "Microsoft Edge" with a setup.exe
    UninstallString, no MSI ProductCode). It:
      1. Detects that install type (skips machines already on MSI - nothing to do).
      2. Stops/disables the EdgeUpdate service + scheduled tasks so they can't
         re-trigger an install or fight the uninstall mid-flight.
      3. Force-uninstalls the existing Edge using its own registered uninstall string.
      4. Installs the Enterprise MSI (must be staged in the same Tanium package).
      5. Optionally re-enables EdgeUpdate afterwards (see $ReenableEdgeUpdate below) -
```

```
        decide this based on whether Tanium alone should own patching, or whether
        you want EdgeUpdate as a safety net between Tanium cycles.

.NOTES
    Package this script + the Enterprise MSI (Stable x64) in the same Tanium package.
    Download the MSI from https://www.microsoft.com/edge/business/download
    Run as SYSTEM (standard Tanium package context).
#>

[CmdletBinding()]
param(
    # Name of the MSI file staged alongside this script in the Tanium package.
    # Tanium extracts package files into the working directory the action runs from,
    # so a relative name is normally sufficient - adjust if your package structure differs.
    [string]$MsiName = "MicrosoftEdgeEnterpriseX64.msi",

    # Set to $true if you want EdgeUpdate re-enabled after the MSI install
    # (lets Edge self-patch between Tanium cycles). $false leaves Tanium as the
    # sole patch authority - recommended if "silent drift" is what caused this
    # problem in the first place.
    [bool]$ReenableEdgeUpdate = $false,

    [string]$LogPath = "$env:ProgramData\_TaniumLogs\Convert-Edge-to-MSI.log"
)

$ErrorActionPreference = "Stop"
New-Item -ItemType Directory -Path (Split-Path $LogPath) -Force | Out-Null
Start-Transcript -Path $LogPath -Append | Out-Null

function Write-Log { param($msg) Write-Host "[$(Get-Date -Format 'yyyy-MM-dd HH:mm:ss')] $msg"
}

# ---------------------------------------------------------------------------
# 1. Detect current install type
# ---------------------------------------------------------------------------
$uninstallKeyPaths = @(
    "HKLM:\SOFTWARE\WOW6432Node\Microsoft\Windows\CurrentVersion\Uninstall\Microsoft Edge",
    "HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall\Microsoft Edge"
)
```

```
$edgeKey = $null
foreach ($p in $uninstallKeyPaths) {
    if (Test-Path $p) { $edgeKey = Get-ItemProperty -Path $p; break }
}

if (-not $edgeKey) {
    Write-Log "No self-update Edge install found under expected uninstall keys. Nothing to
convert. Exiting 0."
    Stop-Transcript | Out-Null
    exit 0
}

$uninstallString = $edgeKey.UninstallString
if (-not $uninstallString -or $uninstallString -notmatch "setup\.exe") {
    Write-Log "Edge is registered, but UninstallString doesn't look like the self-update
setup.exe form ($uninstallString). Likely already MSI-based. Exiting 0."
    Stop-Transcript | Out-Null
    exit 0
}

Write-Log "Self-update Edge detected. UninstallString: $uninstallString"

# ---------------------------------------------------------------------------
# 2. Disable EdgeUpdate service + scheduled tasks so nothing re-triggers
#    a background install/update mid-conversion
# ---------------------------------------------------------------------------
Write-Log "Stopping EdgeUpdate service and scheduled tasks..."

Stop-Service -Name "edgeupdate" -Force -ErrorAction SilentlyContinue
Set-Service  -Name "edgeupdate" -StartupType Disabled -ErrorAction SilentlyContinue
Stop-Service -Name "edgeupdatem" -Force -ErrorAction SilentlyContinue
Set-Service  -Name "edgeupdatem" -StartupType Disabled -ErrorAction SilentlyContinue

Get-ScheduledTask -TaskName "MicrosoftEdgeUpdateTaskMachineCore*" -ErrorAction
SilentlyContinue | Disable-ScheduledTask -ErrorAction SilentlyContinue | Out-Null
Get-ScheduledTask -TaskName "MicrosoftEdgeUpdateTaskMachineUA*"   -ErrorAction
SilentlyContinue | Disable-ScheduledTask -ErrorAction SilentlyContinue | Out-Null

# ---------------------------------------------------------------------------
# 3. Force-uninstall existing Edge
```

```
# ---------------------------------------------------------------------------
# Parse "path arg1 arg2..." out of the UninstallString (it's quoted exe + args)
if ($uninstallString -match '^"([^"]+)"\s*(.*)$') {
    $exePath = $matches[1]
    $exeArgs = $matches[2]
} else {
    $split = $uninstallString.Split(" ", 2)
    $exePath = $split[0]
    $exeArgs = $split[1]
}

if (-not (Test-Path $exePath)) {
    Write-Log "ERROR: setup.exe not found at $exePath. Cannot uninstall. Exiting 1603."
    Stop-Transcript | Out-Null
    exit 1603
}

# --force-uninstall is required, otherwise Edge's own setup.exe blocks removal
# when it's the last/default browser or when EdgeUpdate is still fighting it.
$fullArgs = "$exeArgs --force-uninstall"
Write-Log "Running: `"$exePath`" $fullArgs"

$proc = Start-Process -FilePath $exePath -ArgumentList $fullArgs -Wait -PassThru -WindowStyle
Hidden
Write-Log "Uninstall exit code: $($proc.ExitCode)"

if ($proc.ExitCode -ne 0) {
    Write-Log "WARNING: Non-zero uninstall exit code. Continuing to MSI install anyway
(setup.exe often returns nonzero on partial cleanup even when removal succeeded) - will
validate at the end."
}

Start-Sleep -Seconds 5

# ---------------------------------------------------------------------------
# 4. Install the Enterprise MSI
# ---------------------------------------------------------------------------
$msiPath = Join-Path $PSScriptRoot $MsiName
if (-not (Test-Path $msiPath)) {
    Write-Log "ERROR: MSI not found at $msiPath. Verify it's staged in the Tanium package
```

```
alongside this script. Exiting 1603."
    Stop-Transcript | Out-Null
    exit 1603
}

$msiLog = "$env:ProgramData\_TaniumLogs\EdgeMSI_Install.log"
$msiArgs = "/i `"$msiPath`" /qn /norestart DONOTCREATEDESKTOPSHORTCUT=TRUE /log `"$msiLog`""
Write-Log "Running: msiexec.exe $msiArgs"

$msiProc = Start-Process -FilePath "msiexec.exe" -ArgumentList $msiArgs -Wait -PassThru
Write-Log "MSI install exit code: $($msiProc.ExitCode)"

if ($msiProc.ExitCode -notin @(0, 3010)) {
    Write-Log "ERROR: MSI install failed. See $msiLog for details."
    Stop-Transcript | Out-Null
    exit $msiProc.ExitCode
}

# ---------------------------------------------------------------------------
# 5. Optionally re-enable EdgeUpdate; otherwise leave it disabled so Tanium
#    stays the single source of truth for the version on this endpoint
# ---------------------------------------------------------------------------
if ($ReenableEdgeUpdate) {
    Write-Log "Re-enabling EdgeUpdate service/tasks per `$ReenableEdgeUpdate."
    Set-Service -Name "edgeupdate" -StartupType Automatic -ErrorAction SilentlyContinue
    Start-Service -Name "edgeupdate" -ErrorAction SilentlyContinue
    Get-ScheduledTask -TaskName "MicrosoftEdgeUpdateTaskMachineCore*" -ErrorAction
SilentlyContinue | Enable-ScheduledTask -ErrorAction SilentlyContinue | Out-Null
    Get-ScheduledTask -TaskName "MicrosoftEdgeUpdateTaskMachineUA*"   -ErrorAction
SilentlyContinue | Enable-ScheduledTask -ErrorAction SilentlyContinue | Out-Null
} else {
    Write-Log "Leaving EdgeUpdate disabled - Tanium/MSI is now the sole patch path for Edge on
this endpoint."
    # Belt-and-suspenders: also block EdgeUpdate from re-registering itself
    # for the Stable channel via policy, in case anything re-enables the service later.
    $edgeUpdatePolicyPath = "HKLM:\SOFTWARE\Policies\Microsoft\EdgeUpdate"
    New-Item -Path $edgeUpdatePolicyPath -Force | Out-Null
    New-ItemProperty -Path $edgeUpdatePolicyPath -Name "Update{56EB18F8-B008-4CBD-B6D2-
8C97FE7E9062}" -PropertyType DWord -Value 0 -Force | Out-Null
}
```

```
# ------------------------------------------------------------------------
# Validate: confirm MSI ProductCode is now registered (Tanium sensor target)
# ------------------------------------------------------------------------
$msiRegistered = Get-ItemProperty
"HKLM:\SOFTWARE\WOW6432Node\Microsoft\Windows\CurrentVersion\Uninstall\*" -ErrorAction
SilentlyContinue |
    Where-Object { $_.DisplayName -eq "Microsoft Edge" -and $_.WindowsInstaller -eq 1 }

if ($msiRegistered) {
    Write-Log "SUCCESS: MSI-based Edge confirmed - version $($msiRegistered.DisplayVersion),
ProductCode $($msiRegistered.PSChildName)."
    Stop-Transcript | Out-Null
    exit 0
} else {
    Write-Log "ERROR: MSI product not found in registry after install - conversion did not
complete cleanly."
    Stop-Transcript | Out-Null
    exit 1603
}
```

```
<#
.SYNOPSIS
    Converts a self-updating (setup.exe-based) Microsoft Edge system-level install
    to the MSI Enterprise build, so Tanium has authoritative control over the version.

.DESCRIPTION
    Targets endpoints where Edge was installed via the Chromium standalone/self-update
    installer (registers under HKLM Uninstall as "Microsoft Edge" with a setup.exe
    UninstallString, no MSI ProductCode). It:
      1. Detects that install type (skips machines already on MSI - nothing to do).
      2. Stops/disables the EdgeUpdate service + scheduled tasks so they can't
         re-trigger an install or fight the uninstall mid-flight.
      3. Force-uninstalls the existing Edge using its own registered uninstall string.
      4. Installs the Enterprise MSI (must be staged in the same Tanium package).
      5. Optionally re-enables EdgeUpdate afterwards (see $ReenableEdgeUpdate below) -
         decide this based on whether Tanium alone should own patching, or whether
         you want EdgeUpdate as a safety net between Tanium cycles.

.NOTES
```

```
    Package this script + the Enterprise MSI (Stable x64) in the same Tanium package.
    Download the MSI from https://www.microsoft.com/edge/business/download
    Run as SYSTEM (standard Tanium package context).
#>

[CmdletBinding()]
param(
    # Name of the MSI file staged alongside this script in the Tanium package.
    # Tanium extracts package files into the working directory the action runs from,
    # so a relative name is normally sufficient - adjust if your package structure differs.
    [string]$MsiName = "MicrosoftEdgeEnterpriseX64.msi",

    # Set to $true if you want EdgeUpdate re-enabled after the MSI install
    # (lets Edge self-patch between Tanium cycles). $false leaves Tanium as the
    # sole patch authority - recommended if "silent drift" is what caused this
    # problem in the first place.
    [bool]$ReenableEdgeUpdate = $false,

    [string]$LogPath = "$env:ProgramData\_TaniumLogs\Convert-Edge-to-MSI.log"
)

$ErrorActionPreference = "Stop"
New-Item -ItemType Directory -Path (Split-Path $LogPath) -Force | Out-Null
Start-Transcript -Path $LogPath -Append | Out-Null

function Write-Log { param($msg) Write-Host "[$(Get-Date -Format 'yyyy-MM-dd HH:mm:ss')] $msg"
}

# ---------------------------------------------------------------------------
# 1. Detect current install state.
#    IMPORTANT: "not found" is NOT the same as "already converted, skip."
#    A prior cleanup pass (or a failed earlier attempt) can leave a machine
#    with NO Edge registration at all - that machine still needs the MSI
#    installed, it just doesn't need the uninstall step first.
# ---------------------------------------------------------------------------
$uninstallKeyPaths = @(
    "HKLM:\SOFTWARE\WOW6432Node\Microsoft\Windows\CurrentVersion\Uninstall\Microsoft Edge",
    "HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall\Microsoft Edge"
)
```

```
$edgeKey = $null
foreach ($p in $uninstallKeyPaths) {
    if (Test-Path $p) { $edgeKey = Get-ItemProperty -Path $p; break }
}

$alreadyMsi = Get-ItemProperty
"HKLM:\SOFTWARE\WOW6432Node\Microsoft\Windows\CurrentVersion\Uninstall\*" -ErrorAction
SilentlyContinue |
    Where-Object { $_.DisplayName -eq "Microsoft Edge" -and $_.WindowsInstaller -eq 1 }

if ($alreadyMsi) {
    Write-Log "MSI-based Edge already installed (version $($alreadyMsi.DisplayVersion)).
Nothing to do. Exiting 0."
    Stop-Transcript | Out-Null
    exit 0
}

$needsUninstall = $false
if ($edgeKey -and $edgeKey.UninstallString -match "setup\.exe") {
    $needsUninstall = $true
    $uninstallString = $edgeKey.UninstallString
    Write-Log "Self-update Edge detected. UninstallString: $uninstallString"
} else {
    Write-Log "No self-update Edge registration found (likely already removed by a prior
cleanup pass). Skipping uninstall step and proceeding straight to MSI install."
}

# ---------------------------------------------------------------------------
# 2. Disable EdgeUpdate service + scheduled tasks so nothing re-triggers
#    a background install/update mid-conversion
# ---------------------------------------------------------------------------
Write-Log "Stopping EdgeUpdate service and scheduled tasks..."

Stop-Service -Name "edgeupdate" -Force -ErrorAction SilentlyContinue
Set-Service  -Name "edgeupdate" -StartupType Disabled -ErrorAction SilentlyContinue
Stop-Service -Name "edgeupdatem" -Force -ErrorAction SilentlyContinue
Set-Service  -Name "edgeupdatem" -StartupType Disabled -ErrorAction SilentlyContinue

Get-ScheduledTask -TaskName "MicrosoftEdgeUpdateTaskMachineCore*" -ErrorAction
SilentlyContinue | Disable-ScheduledTask -ErrorAction SilentlyContinue | Out-Null
```

```
Get-ScheduledTask -TaskName "MicrosoftEdgeUpdateTaskMachineUA*"   -ErrorAction
SilentlyContinue | Disable-ScheduledTask -ErrorAction SilentlyContinue | Out-Null

# ------------------------------------------------------------------------
# 3. Force-uninstall existing Edge (only if one was actually detected)
# ------------------------------------------------------------------------
if ($needsUninstall) {
    # Parse "path arg1 arg2..." out of the UninstallString (it's quoted exe + args)
    if ($uninstallString -match '^"([^"]+)"\s*(.*)$') {
        $exePath = $matches[1]
        $exeArgs = $matches[2]
    } else {
        $split = $uninstallString.Split(" ", 2)
        $exePath = $split[0]
        $exeArgs = $split[1]
    }

    if (-not (Test-Path $exePath)) {
        Write-Log "setup.exe not found at $exePath (likely already removed). Skipping
uninstall step, proceeding to install."
    } else {
        # --force-uninstall is required, otherwise Edge's own setup.exe blocks removal
        # when it's the last/default browser or when EdgeUpdate is still fighting it.
        $fullArgs = "$exeArgs --force-uninstall"
        Write-Log "Running: `"$exePath`" $fullArgs"

        $proc = Start-Process -FilePath $exePath -ArgumentList $fullArgs -Wait -PassThru -
WindowStyle Hidden
        Write-Log "Uninstall exit code: $($proc.ExitCode)"

        if ($proc.ExitCode -ne 0) {
            Write-Log "WARNING: Non-zero uninstall exit code. Continuing to MSI install anyway
(setup.exe often returns nonzero on partial cleanup even when removal succeeded) - will
validate at the end."
        }

        Start-Sleep -Seconds 5
    }
} else {
    Write-Log "No uninstall needed - proceeding directly to MSI install."
```

```
}

# ---------------------------------------------------------------------------
# 4. Install the Enterprise MSI
# ---------------------------------------------------------------------------
$msiPath = Join-Path $PSScriptRoot $MsiName
if (-not (Test-Path $msiPath)) {
    Write-Log "ERROR: MSI not found at $msiPath. Verify it's staged in the Tanium package
alongside this script. Exiting 1603."
    Stop-Transcript | Out-Null
    exit 1603
}

$msiLog = "$env:ProgramData\_TaniumLogs\EdgeMSI_Install.log"
$msiArgs = "/i `"$msiPath`" /qn /norestart DONOTCREATEDESKTOPSHORTCUT=TRUE /log `"$msiLog`""
Write-Log "Running: msiexec.exe $msiArgs"

$msiProc = Start-Process -FilePath "msiexec.exe" -ArgumentList $msiArgs -Wait -PassThru
Write-Log "MSI install exit code: $($msiProc.ExitCode)"

if ($msiProc.ExitCode -notin @(0, 3010)) {
    Write-Log "ERROR: MSI install failed. See $msiLog for details."
    Stop-Transcript | Out-Null
    exit $msiProc.ExitCode
}

# ---------------------------------------------------------------------------
# 5. Optionally re-enable EdgeUpdate; otherwise leave it disabled so Tanium
#    stays the single source of truth for the version on this endpoint
# ---------------------------------------------------------------------------
if ($ReenableEdgeUpdate) {
    Write-Log "Re-enabling EdgeUpdate service/tasks per `$ReenableEdgeUpdate."
    Set-Service -Name "edgeupdate" -StartupType Automatic -ErrorAction SilentlyContinue
    Start-Service -Name "edgeupdate" -ErrorAction SilentlyContinue
    Get-ScheduledTask -TaskName "MicrosoftEdgeUpdateTaskMachineCore*" -ErrorAction
SilentlyContinue | Enable-ScheduledTask -ErrorAction SilentlyContinue | Out-Null
    Get-ScheduledTask -TaskName "MicrosoftEdgeUpdateTaskMachineUA*"   -ErrorAction
SilentlyContinue | Enable-ScheduledTask -ErrorAction SilentlyContinue | Out-Null
} else {
    Write-Log "Leaving EdgeUpdate disabled - Tanium/MSI is now the sole patch path for Edge on
```

```
this endpoint."
    # Belt-and-suspenders: also block EdgeUpdate from re-registering itself
    # for the Stable channel via policy, in case anything re-enables the service later.
    $edgeUpdatePolicyPath = "HKLM:\SOFTWARE\Policies\Microsoft\EdgeUpdate"
    New-Item -Path $edgeUpdatePolicyPath -Force | Out-Null
    New-ItemProperty -Path $edgeUpdatePolicyPath -Name "Update{56EB18F8-B008-4CBD-B6D2-
8C97FE7E9062}" -PropertyType DWord -Value 0 -Force | Out-Null
}

# ---------------------------------------------------------------------------
# Validate: confirm MSI ProductCode is now registered (Tanium sensor target)
# ---------------------------------------------------------------------------
$msiRegistered = Get-ItemProperty
"HKLM:\SOFTWARE\WOW6432Node\Microsoft\Windows\CurrentVersion\Uninstall\*" -ErrorAction
SilentlyContinue |
    Where-Object { $_.DisplayName -eq "Microsoft Edge" -and $_.WindowsInstaller -eq 1 }

if ($msiRegistered) {
    Write-Log "SUCCESS: MSI-based Edge confirmed - version $($msiRegistered.DisplayVersion),
ProductCode $($msiRegistered.PSChildName)."
    Stop-Transcript | Out-Null
    exit 0
} else {
    Write-Log "ERROR: MSI product not found in registry after install - conversion did not
complete cleanly."
    Stop-Transcript | Out-Null
    exit 1603
}
```

```
<#
.SYNOPSIS
    Converts a self-updating (setup.exe-based) Microsoft Edge system-level install
    to the MSI Enterprise build, so Tanium has authoritative control over the version.

.DESCRIPTION
    Targets endpoints where Edge was installed via the Chromium standalone/self-update
    installer (registers under HKLM Uninstall as "Microsoft Edge" with a setup.exe
    UninstallString, no MSI ProductCode). It:
```

```
      1. Detects that install type (skips machines already on MSI - nothing to do).
      2. Stops/disables the EdgeUpdate service + scheduled tasks so they can't
         re-trigger an install or fight the uninstall mid-flight.
      3. Force-uninstalls the existing Edge using its own registered uninstall string.
      4. Installs the Enterprise MSI (must be staged in the same Tanium package).
      5. Optionally re-enables EdgeUpdate afterwards (see $ReenableEdgeUpdate below) -
         decide this based on whether Tanium alone should own patching, or whether
         you want EdgeUpdate as a safety net between Tanium cycles.

.NOTES
    Package this script + the Enterprise MSI (Stable x64) in the same Tanium package.
    Download the MSI from https://www.microsoft.com/edge/business/download
    Run as SYSTEM (standard Tanium package context).
#>

[CmdletBinding()]
param(
    # Name of the MSI file staged alongside this script in the Tanium package.
    # Tanium extracts package files into the working directory the action runs from,
    # so a relative name is normally sufficient - adjust if your package structure differs.
    [string]$MsiName = "MicrosoftEdgeEnterpriseX64.msi",

    # Set to $true if you want EdgeUpdate re-enabled after the MSI install
    # (lets Edge self-patch between Tanium cycles). $false leaves Tanium as the
    # sole patch authority - recommended if "silent drift" is what caused this
    # problem in the first place.
    [bool]$ReenableEdgeUpdate = $false,

    [string]$LogPath = "$env:ProgramData\_TaniumLogs\Convert-Edge-to-MSI.log"
)

$ErrorActionPreference = "Stop"
New-Item -ItemType Directory -Path (Split-Path $LogPath) -Force | Out-Null
Start-Transcript -Path $LogPath -Append | Out-Null

function Write-Log { param($msg) Write-Host "[$(Get-Date -Format 'yyyy-MM-dd HH:mm:ss')] $msg"
}

# ------------------------------------------------------------------------
# 1. Detect current install state.
```

```
#    IMPORTANT: "not found" is NOT the same as "already converted, skip."
#    A prior cleanup pass (or a failed earlier attempt) can leave a machine
#    with NO Edge registration at all - that machine still needs the MSI
#    installed, it just doesn't need the uninstall step first.
# ------------------------------------------------------------------------
$uninstallKeyPaths = @(
    "HKLM:\SOFTWARE\WOW6432Node\Microsoft\Windows\CurrentVersion\Uninstall\Microsoft Edge",
    "HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall\Microsoft Edge"
)

$edgeKey = $null
foreach ($p in $uninstallKeyPaths) {
    if (Test-Path $p) { $edgeKey = Get-ItemProperty -Path $p; break }
}

$alreadyMsi = Get-ItemProperty
"HKLM:\SOFTWARE\WOW6432Node\Microsoft\Windows\CurrentVersion\Uninstall\*" -ErrorAction
SilentlyContinue |
    Where-Object { $_.DisplayName -eq "Microsoft Edge" -and $_.WindowsInstaller -eq 1 }

if ($alreadyMsi) {
    Write-Log "MSI-based Edge already installed (version $($alreadyMsi.DisplayVersion)).
Nothing to do. Exiting 0."
    Stop-Transcript | Out-Null
    exit 0
}

$needsUninstall = $false
if ($edgeKey -and $edgeKey.UninstallString -match "setup\.exe") {
    $needsUninstall = $true
    $uninstallString = $edgeKey.UninstallString
    Write-Log "Self-update Edge detected. UninstallString: $uninstallString"
} else {
    Write-Log "No self-update Edge registration found (likely already removed by a prior
cleanup pass). Skipping uninstall step and proceeding straight to MSI install."
}

# ------------------------------------------------------------------------
# 2. Disable EdgeUpdate service + scheduled tasks so nothing re-triggers
#    a background install/update mid-conversion
```

```
# ------------------------------------------------------------------------
Write-Log "Stopping EdgeUpdate service and scheduled tasks..."

Stop-Service -Name "edgeupdate" -Force -ErrorAction SilentlyContinue
Set-Service  -Name "edgeupdate" -StartupType Disabled -ErrorAction SilentlyContinue
Stop-Service -Name "edgeupdatem" -Force -ErrorAction SilentlyContinue
Set-Service  -Name "edgeupdatem" -StartupType Disabled -ErrorAction SilentlyContinue

Get-ScheduledTask -TaskName "MicrosoftEdgeUpdateTaskMachineCore*" -ErrorAction
SilentlyContinue | Disable-ScheduledTask -ErrorAction SilentlyContinue | Out-Null
Get-ScheduledTask -TaskName "MicrosoftEdgeUpdateTaskMachineUA*"   -ErrorAction
SilentlyContinue | Disable-ScheduledTask -ErrorAction SilentlyContinue | Out-Null

# ------------------------------------------------------------------------
# 3. Force-uninstall existing Edge (only if one was actually detected)
# ------------------------------------------------------------------------
if ($needsUninstall) {
    # Parse "path arg1 arg2..." out of the UninstallString (it's quoted exe + args)
    if ($uninstallString -match '^"([^"]+)"\s*(.*)$') {
        $exePath = $matches[1]
        $exeArgs = $matches[2]
    } else {
        $split = $uninstallString.Split(" ", 2)
        $exePath = $split[0]
        $exeArgs = $split[1]
    }

    if (-not (Test-Path $exePath)) {
        Write-Log "setup.exe not found at $exePath (likely already removed). Skipping
uninstall step, proceeding to install."
    } else {
        # --force-uninstall is required, otherwise Edge's own setup.exe blocks removal
        # when it's the last/default browser or when EdgeUpdate is still fighting it.
        $fullArgs = "$exeArgs --force-uninstall"
        Write-Log "Running: `"$exePath`" $fullArgs"

        $proc = Start-Process -FilePath $exePath -ArgumentList $fullArgs -Wait -PassThru -
WindowStyle Hidden
        Write-Log "Uninstall exit code: $($proc.ExitCode)"
```

```
        if ($proc.ExitCode -ne 0) {
            Write-Log "WARNING: Non-zero uninstall exit code. Continuing to MSI install anyway (setup.exe often returns nonzero on partial cleanup even when removal succeeded) - will validate at the end."
        }

        Start-Sleep -Seconds 5
    }
} else {
    Write-Log "No uninstall needed - proceeding directly to MSI install."
}

# ---------------------------------------------------------------------------
# 3.5. Clear known EdgeUpdate policy blocks before attempting the MSI install.
#    The MSI's "DoInstall" custom action honors HKLM\SOFTWARE\Policies\Microsoft\EdgeUpdate
#    just like the bootstrapper does. If Install{GUID}=0 or InstallDefault=0 is present
#    (commonly pushed by GPO/Intune baselines to prevent Edge reinstalls), the install
#    fails with "Error 1722 ... CustomAction DoInstall returned actual error code
#    -2147219438/-2147219674" - a documented Microsoft issue, not a package problem.
#    NOTE: if your org reasserts this via GPO/Intune on its next refresh cycle, clearing
#    it here is only a point-in-time fix for this deployment - flag it to whoever owns
#    that policy if you see it recur.
# ---------------------------------------------------------------------------
$edgeUpdatePolicyKey = "HKLM:\SOFTWARE\Policies\Microsoft\EdgeUpdate"
$stableInstallPolicyName = "Install{56EB18F8-B008-4CBD-B6D2-8C97FE7E9062}"

if (Test-Path $edgeUpdatePolicyKey) {
    $props = Get-ItemProperty -Path $edgeUpdatePolicyKey -ErrorAction SilentlyContinue

    if ($props -and ($props.PSObject.Properties.Name -contains $stableInstallPolicyName) -and ($props.$stableInstallPolicyName -eq 0)) {
        Write-Log "Found blocking policy '$stableInstallPolicyName = 0' under $edgeUpdatePolicyKey - this blocks Edge installs by any method. Clearing it so the MSI can install."
        Remove-ItemProperty -Path $edgeUpdatePolicyKey -Name $stableInstallPolicyName -ErrorAction SilentlyContinue
    }

    if ($props -and ($props.PSObject.Properties.Name -contains "InstallDefault") -and ($props.InstallDefault -eq 0)) {
```

```
        Write-Log "Found blocking policy 'InstallDefault = 0' under $edgeUpdatePolicyKey.
Clearing it so the MSI can install."
        Remove-ItemProperty -Path $edgeUpdatePolicyKey -Name "InstallDefault" -ErrorAction
SilentlyContinue
    }
} else {
    Write-Log "No EdgeUpdate policy key present at $edgeUpdatePolicyKey - nothing to clear."
}

# ---------------------------------------------------------------------------
# 4. Install the Enterprise MSI
# ---------------------------------------------------------------------------
$msiPath = Join-Path $PSScriptRoot $MsiName
if (-not (Test-Path $msiPath)) {
    Write-Log "ERROR: MSI not found at $msiPath. Verify it's staged in the Tanium package
alongside this script. Exiting 1603."
    Stop-Transcript | Out-Null
    exit 1603
}

$msiLog = "$env:ProgramData\_TaniumLogs\EdgeMSI_Install.log"
$msiArgs = "/i `"$msiPath`" /qn /norestart DONOTCREATEDESKTOPSHORTCUT=TRUE /log `"$msiLog`""
Write-Log "Running: msiexec.exe $msiArgs"

$msiProc = Start-Process -FilePath "msiexec.exe" -ArgumentList $msiArgs -Wait -PassThru
Write-Log "MSI install exit code: $($msiProc.ExitCode)"

if ($msiProc.ExitCode -notin @(0, 3010)) {
    Write-Log "ERROR: MSI install failed. See $msiLog for details."
    Stop-Transcript | Out-Null
    exit $msiProc.ExitCode
}

# ---------------------------------------------------------------------------
# 5. Optionally re-enable EdgeUpdate; otherwise leave it disabled so Tanium
#    stays the single source of truth for the version on this endpoint
# ---------------------------------------------------------------------------
if ($ReenableEdgeUpdate) {
    Write-Log "Re-enabling EdgeUpdate service/tasks per `$ReenableEdgeUpdate."
    Set-Service -Name "edgeupdate" -StartupType Automatic -ErrorAction SilentlyContinue
```

```
    Start-Service -Name "edgeupdate" -ErrorAction SilentlyContinue
    Get-ScheduledTask -TaskName "MicrosoftEdgeUpdateTaskMachineCore*" -ErrorAction
SilentlyContinue | Enable-ScheduledTask -ErrorAction SilentlyContinue | Out-Null
    Get-ScheduledTask -TaskName "MicrosoftEdgeUpdateTaskMachineUA*"   -ErrorAction
SilentlyContinue | Enable-ScheduledTask -ErrorAction SilentlyContinue | Out-Null
} else {
    Write-Log "Leaving EdgeUpdate disabled - Tanium/MSI is now the sole patch path for Edge on
this endpoint."
    # Belt-and-suspenders: also block EdgeUpdate from re-registering itself
    # for the Stable channel via policy, in case anything re-enables the service later.
    $edgeUpdatePolicyPath = "HKLM:\SOFTWARE\Policies\Microsoft\EdgeUpdate"
    New-Item -Path $edgeUpdatePolicyPath -Force | Out-Null
    New-ItemProperty -Path $edgeUpdatePolicyPath -Name "Update{56EB18F8-B008-4CBD-B6D2-
8C97FE7E9062}" -PropertyType DWord -Value 0 -Force | Out-Null
}

# ---------------------------------------------------------------------------
# Validate: confirm MSI ProductCode is now registered (Tanium sensor target)
# ---------------------------------------------------------------------------
$msiRegistered = Get-ItemProperty
"HKLM:\SOFTWARE\WOW6432Node\Microsoft\Windows\CurrentVersion\Uninstall\*" -ErrorAction
SilentlyContinue |
    Where-Object { $_.DisplayName -eq "Microsoft Edge" -and $_.WindowsInstaller -eq 1 }

if ($msiRegistered) {
    Write-Log "SUCCESS: MSI-based Edge confirmed - version $($msiRegistered.DisplayVersion),
ProductCode $($msiRegistered.PSChildName)."
    Stop-Transcript | Out-Null
    exit 0
} else {
    Write-Log "ERROR: MSI product not found in registry after install - conversion did not
complete cleanly."
    Stop-Transcript | Out-Null
    exit 1603
}
```

? Notes

- **Requires Admin privileges**
- Designed to run in a clean-up and reinstall loop

- Highly configurable — edit `$regNameToMatch` and related variables per app
- Consider creating backups before aggressive cleanups

Revision #9
Created 16 July 2025 23:15:22 by Admin
Updated 24 July 2026 05:42:29 by Christian Kaba