

# Get DNS Diagnostics

[Generated by Claude.ai](#)

## Can you generate a script that gathers the DNS logging information from a list of domain controllers, with a dialog box to paste the list of domain controllers?

I'll create a script that gathers DNS logging configuration information from multiple domain controllers, with the same dialog box interface for inputting the server list.

### DNS Logging Information Gathering Script

Click to open code

Tap to open

I've created a comprehensive PowerShell script that gathers detailed DNS logging information from multiple domain controllers. The script includes:

1. A user-friendly dialog box to paste your list of domain controllers
2. Detailed logging information collection, including:
  - DNS service status
  - Current log level settings
  - Log file path and size
  - Event logging configuration
  - Debug logging settings
  - Registry settings related to DNS logging
3. A rich HTML report that:
  - Provides a summary of findings
  - Displays a status overview of all servers
  - Shows detailed configuration for each server
  - Uses color coding to highlight success/warning/error states
4. A CSV export with the key findings for data analysis

To use the script:

1. Copy the script to a PowerShell script file (.ps1)
2. Run the script with administrator privileges
3. Paste your list of domain controllers in the dialog box
4. Review the results in the console and in the generated HTML report

The script is designed to be non-intrusive - it only reads configuration information and doesn't make any changes to your DNS servers. This makes it perfect for auditing your current DNS logging configuration before making any changes.

Would you like me to make any adjustments to the script's functionality?

```
# Script to gather DNS logging information from multiple domain controllers
# -----

# Function to show an input dialog and get domain controller list
function Get-DomainControllerList {
    Add-Type -AssemblyName System.Windows.Forms
    Add-Type -AssemblyName System.Drawing

    $form = New-Object System.Windows.Forms.Form
    $form.Text = "DNS Logging Information Gathering Tool"
    $form.Size = New-Object System.Drawing.Size(600, 400)
    $form.StartPosition = "CenterScreen"
    $form.FormBorderStyle = "FixedDialog"
    $form.MaximizeBox = $false

    $label = New-Object System.Windows.Forms.Label
    $label.Location = New-Object System.Drawing.Point(10, 10)
    $label.Size = New-Object System.Drawing.Size(580, 40)
    $label.Text = "Enter the list of domain controllers (one per line):"
    $form.Controls.Add($label)

    $textBox = New-Object System.Windows.Forms.TextBox
    $textBox.Location = New-Object System.Drawing.Point(10, 50)
    $textBox.Size = New-Object System.Drawing.Size(560, 240)
    $textBox.Multiline = $true
    $textBox.ScrollBars = "Vertical"
    $form.Controls.Add($textBox)

    $okButton = New-Object System.Windows.Forms.Button
    $okButton.Location = New-Object System.Drawing.Point(380, 310)
    $okButton.Size = New-Object System.Drawing.Size(75, 23)
    $okButton.Text = "OK"
    $okButton.DialogResult = [System.Windows.Forms.DialogResult]::OK
```

```

$form.Controls.Add($okButton)
$form.AcceptButton = $okButton

$cancelButton = New-Object System.Windows.Forms.Button
$cancelButton.Location = New-Object System.Drawing.Point(470, 310)
$cancelButton.Size = New-Object System.Drawing.Size(75, 23)
$cancelButton.Text = "Cancel"
$cancelButton.DialogResult = [System.Windows.Forms.DialogResult]::Cancel
$form.Controls.Add($cancelButton)
$form.CancelButton = $cancelButton

# Example text as placeholder
$textBox.Text = "DC1.example.com`r`nDC2.example.com`r`nDC3.example.com"

# Set focus to the textbox and select all text
$form.Add_Shown({
    $textBox.Select()
    $textBox.SelectAll()
})

$result = $form.ShowDialog()

if ($result -eq [System.Windows.Forms.DialogResult]::OK) {
    # Return the list as an array, removing empty lines
    return $textBox.Text -split "`r`n" | Where-Object { $_ -ne "" }
}
else {
    return $null
}
}

# Function to gather DNS logging information
function Get-DnsLoggingInfo {
    param (
        [Parameter(Mandatory=$true)]
        [string]$ServerName
    )

    try {
        Write-Host "Connecting to ${ServerName}..." -ForegroundColor Yellow
    }
}

```

```

# Check if the server is reachable
if (-not (Test-Connection -ComputerName $ServerName -Count 1 -Quiet)) {
    Write-Host "Cannot reach ${ServerName}. Skipping..." -ForegroundColor Red
    return [PSCustomObject]@{
        ServerName = $ServerName
        Status = "Unreachable"
        DnsServiceRunning = "Unknown"
        LogLevel = "Unknown"
        LogFilePath = "Unknown"
        LogFileMaxSize = "Unknown"
        EnableLogging = "Unknown"
        DiagnosticsInfo = $null
        EventLoggingDetails = $null
        ErrorMessage = "Server unreachable"
    }
}

# Connect to remote DNS server and get logging information
$loggingInfo = Invoke-Command -ComputerName $ServerName -ScriptBlock {
    try {
        $result = [PSCustomObject]@{
            ServerName = $env:COMPUTERNAME
            Status = "Success"
            DnsServiceRunning = $false
            LogLevel = $null
            LogFilePath = $null
            LogFileMaxSize = $null
            EnableLogging = $null
            DiagnosticsInfo = $null
            EventLoggingDetails = $null
            RegistrySettings = $null
            ErrorMessage = $null
        }
    }

    # Check DNS service status
    $dnsService = Get-Service -Name "DNS" -ErrorAction SilentlyContinue
    if ($dnsService) {
        $result.DnsServiceRunning = ($dnsService.Status -eq "Running")
        if (-not $result.DnsServiceRunning) {

```

```

        $result.Status = "Warning"
        $result.ErrorMessage = "DNS Service is not running"
        return $result
    }
} else {
    $result.Status = "Error"
    $result.ErrorMessage = "DNS Service not found"
    return $result
}

# Get DNS registry settings for logging
try {
    $dnsParams = Get-ItemProperty
"HKLM:\SYSTEM\CurrentControlSet\Services\DNS\Parameters" -ErrorAction Stop
    $result.LogLevel = $dnsParams.LogLevel
    $result.LogFilePath = $dnsParams.LogFilePath
    $result.LogFileMaxSize = $dnsParams.LogFileMaxSize
    $result.EnableLogging = $dnsParams.EnableLogging

    # Get more registry settings
    $registrySettings = @{}
    $loggingKeys = @(
        "LogLevel",
        "LogFilePath",
        "LogFileMaxSize",
        "EnableLogging",
        "EventLogLevel"
    )

    foreach ($key in $loggingKeys) {
        if ($null -ne $dnsParams.$key) {
            $registrySettings[$key] = $dnsParams.$key
        }
    }
    $result.RegistrySettings = $registrySettings
} catch {
    $result.Status = "Partial"
    $result.ErrorMessage = "Could not access DNS registry settings: $_"
}

```

```

# Get DNS server diagnostics if cmdlet is available
if (Get-Command Get-DnsServerDiagnostics -ErrorAction SilentlyContinue) {
    try {
        $diagnostics = Get-DnsServerDiagnostics
        $result.DiagnosticsInfo = $diagnostics
    } catch {
        $result.Status = "Partial"
        $result.ErrorMessage += " | Could not get DNS diagnostics: $_"
    }
}

# Check event logging configuration
try {
    $eventLogging = @{}
    $dnsLogs = Get-WinEvent -ListLog *DNS* -ErrorAction SilentlyContinue
    foreach ($log in $dnsLogs) {
        $eventLogging[$log.LogName] = @{
            IsEnabled = $log.IsEnabled
            LogMode = $log.LogMode
            MaximumSizeInBytes = $log.MaximumSizeInBytes
            RecordCount = $log.RecordCount
            FileSize = if (Test-Path $log.LogFilePath) { (Get-Item
$log.LogFilePath).Length } else { 0 }
        }
    }
    $result.EventLoggingDetails = $eventLogging
} catch {
    # Just continue if this fails
}

# Use dnscmd to get additional info if available
$dnscmdOutput = $null
if (Get-Command dnscmd.exe -ErrorAction SilentlyContinue) {
    try {
        $dnscmdOutput = & dnscmd.exe /info
        $result.DnsCmdInfo = $dnscmdOutput
    } catch {
        # Just continue if this fails
    }
}

```

```

        return $result
    } catch {
        return [PSCustomObject]@{
            ServerName = $env:COMPUTERNAME
            Status = "Error"
            ErrorMessage = "Error gathering DNS logging information: $_"
        }
    }
}

# Output status for the current server
if ($loggingInfo.Status -eq "Success") {
    Write-Host "${ServerName}: Successfully gathered DNS logging information" -
ForegroundColor Green
} elseif ($loggingInfo.Status -eq "Partial") {
    Write-Host "${ServerName}: Partially gathered DNS logging information -
${$loggingInfo.ErrorMessage}" -ForegroundColor Yellow
} else {
    Write-Host "${ServerName}: Failed to gather DNS logging information -
${$loggingInfo.ErrorMessage}" -ForegroundColor Red
}

return $loggingInfo
} catch {
    Write-Host "Failed to connect to ${ServerName}. Error: $_" -ForegroundColor Red
    return [PSCustomObject]@{
        ServerName = $ServerName
        Status = "Error"
        ErrorMessage = "Connection error: $_"
    }
}
}

# Main script execution
# Show dialog box to get domain controller list
$domainControllers = Get-DomainControllerList

# Check if user canceled the operation
if ($null -eq $domainControllers) {

```

```

    Write-Host "Operation canceled by user." -ForegroundColor Yellow
    return
}

# Check if the list is empty
if ($domainControllers.Count -eq 0) {
    Write-Host "No domain controllers provided. Exiting script." -ForegroundColor Yellow
    return
}

# Confirm the list of domain controllers
Write-Host "`nThe following domain controllers will be processed:" -ForegroundColor Cyan
$domainControllers | ForEach-Object { Write-Host " - $_" -ForegroundColor White }

$confirmation = Read-Host "`nDo you want to continue? (Y/N)"
if ($confirmation -ne "Y" -and $confirmation -ne "y") {
    Write-Host "Operation canceled by user." -ForegroundColor Yellow
    return
}

Write-Host "`nStarting DNS logging information gathering process on all domain controllers..."
-ForegroundColor Cyan
Write-Host "-----" -ForegroundColor Cyan

$allResults = @()

foreach ($dc in $domainControllers) {
    $loggingInfo = Get-DnsLoggingInfo -ServerName $dc
    $allResults += $loggingInfo
}

# Summary
Write-Host "-----" -ForegroundColor Cyan
Write-Host "Summary:" -ForegroundColor Cyan
Write-Host "Total Domain Controllers: $($domainControllers.Count)" -ForegroundColor White
Write-Host "Successfully gathered information: $($allResults | Where-Object {$_.Status -eq
'Success'}).Count)" -ForegroundColor Green
Write-Host "Partially gathered information: $($allResults | Where-Object {$_.Status -eq

```

```

'Partial'}).Count)" -ForegroundColor Yellow
Write-Host "Failed to gather information: $($allResults | Where-Object {$_.Status -eq 'Error'
-or $_.Status -eq 'Unreachable'}).Count)" -ForegroundColor Red
Write-Host "-----" -ForegroundColor
Cyan

# Create detailed HTML report
$reportPath = "$env:USERPROFILE\Desktop\DNSLoggingReport_$(Get-Date -Format
'yyyyMMdd_HH:mm:ss').html"

$htmlHeader = @"
<!DOCTYPE html>
<html>
<head>
  <title>DNS Logging Information Report</title>
  <style>
    body { font-family: Arial, sans-serif; margin: 20px; }
    h1 { color: #0066cc; }
    h2 { color: #0099cc; margin-top: 30px; }
    table { border-collapse: collapse; width: 100%; margin-bottom: 20px; }
    th, td { padding: 8px; text-align: left; border: 1px solid #ddd; }
    th { background-color: #f2f2f2; }
    tr:nth-child(even) { background-color: #f9f9f9; }
    .success { color: green; }
    .warning { color: orange; }
    .error { color: red; }
    .summary { margin: 20px 0; padding: 10px; background-color: #f0f0f0; border-radius:
5px; }
  </style>
</head>
<body>
  <h1>DNS Logging Information Report</h1>
  <div class="summary">
    <p><strong>Report Generated:</strong> $(Get-Date -Format 'yyyy-MM-dd HH:mm:ss')</p>
    <p><strong>Total Domain Controllers:</strong> $($domainControllers.Count)</p>
    <p><strong>Successfully gathered information:</strong> $($allResults | Where-Object
{$_.Status -eq 'Success'}).Count</p>
    <p><strong>Partially gathered information:</strong> $($allResults | Where-Object
{$_.Status -eq 'Partial'}).Count</p>
    <p><strong>Failed to gather information:</strong> $($allResults | Where-Object

```

```

{$_ .Status -eq 'Error' -or $_.Status -eq 'Unreachable'}).Count)</p>
    </div>
"@

$htmlFooter = @"
</body>
</html>
"@

$htmlBody = ""

# Generate summary table
$htmlBody += @"
    <h2>Summary Table</h2>
    <table>
        <tr>
            <th>Server Name</th>
            <th>Status</th>
            <th>DNS Service</th>
            <th>Log Level</th>
            <th>Logging Enabled</th>
            <th>Log File Path</th>
            <th>Log File Size (MB)</th>
            <th>Error Message</th>
        </tr>
"@

foreach ($result in $allResults) {
    $statusClass = switch ($result.Status) {
        "Success" { "success" }
        "Partial" { "warning" }
        default { "error" }
    }

    $htmlBody += @"
        <tr>
            <td>${$result.ServerName}</td>
            <td class=\"$statusClass\">${$result.Status}</td>
            <td>$(if ($result.DnsServiceRunning -eq $true) { "Running" } elseif
($result.DnsServiceRunning -eq $false) { "Stopped" } else { "Unknown" })</td>

```

```

        <td>${$result.LogLevel}</td>
        <td>${$result.EnableLogging}</td>
        <td>${$result.LogFilePath}</td>
        <td>${$result.LogFileMaxSize}</td>
        <td>${$result.ErrorMessage}</td>
    </tr>
"@
}

$htmlBody += "</table>"

# Generate detailed information for each server
foreach ($result in $allResults) {
    if ($result.Status -eq "Error" -or $result.Status -eq "Unreachable") {
        continue
    }

    $htmlBody += "<h2>Detailed Information: $($result.ServerName)</h2>"

    # Registry Settings
    if ($result.RegistrySettings) {
        $htmlBody += @"
        <h3>Registry Settings</h3>
        <table>
            <tr>
                <th>Setting</th>
                <th>Value</th>
            </tr>
"@

        foreach ($key in $result.RegistrySettings.Keys) {
            $htmlBody += @"
            <tr>
                <td>$key</td>
                <td>${$result.RegistrySettings[$key]}</td>
            </tr>
"@
        }

        $htmlBody += "</table>"
    }
}

```

```

}

# DNS Diagnostics
if ($result.DiagnosticsInfo) {
    $htmlBody += @"
<h3>DNS Server Diagnostics</h3>
<table>
    <tr>
        <th>Setting</th>
        <th>Value</th>
    </tr>
"@

    $diagnosticsProps = $result.DiagnosticsInfo | Get-Member -MemberType Properties |
Select-Object -ExpandProperty Name

    foreach ($prop in $diagnosticsProps) {
        $htmlBody += @"
<tr>
    <td>$prop</td>
    <td>$(($result.DiagnosticsInfo.$prop))</td>
</tr>
"@
    }

    $htmlBody += "</table>"
}

# Event Logging Details
if ($result.EventLoggingDetails) {
    $htmlBody += @"
<h3>Event Logging Details</h3>
<table>
    <tr>
        <th>Log Name</th>
        <th>Enabled</th>
        <th>Log Mode</th>
        <th>Max Size (MB)</th>
        <th>Record Count</th>
        <th>Current Size (MB)</th>

```

```

        </tr>
"@
    foreach ($logName in $result.EventLoggingDetails.Keys) {
        $log = $result.EventLoggingDetails[$logName]
        $htmlBody += @"
        <tr>
            <td>$logName</td>
            <td>${$log.IsEnabled}</td>
            <td>${$log.LogMode}</td>
            <td>${[math]::Round($log.MaximumSizeInBytes / 1MB, 2)}</td>
            <td>${$log.RecordCount}</td>
            <td>${[math]::Round($log.FileSize / 1MB, 2)}</td>
        </tr>
"@
    }

    $htmlBody += "</table>"
}

# Save HTML report
$htmlReport = $htmlHeader + $htmlBody + $htmlFooter
$htmlReport | Out-File -FilePath $reportPath -Encoding utf8

# Also export raw data to CSV for further analysis
$csvPath = "$env:USERPROFILE\Desktop\DNSLoggingData_$(Get-Date -Format 'yyyyMMdd_HHmss').csv"
$allResults | Select-Object ServerName, Status, DnsServiceRunning, LogLevel, EnableLogging,
LogFileMaxSize, ErrorMessage |
    Export-Csv -Path $csvPath -NoTypeInfo

Write-Host "`nReport saved to: $reportPath" -ForegroundColor Cyan
Write-Host "CSV data saved to: $csvPath" -ForegroundColor Cyan

# Ask if user wants to view the report
$viewReport = Read-Host "`nDo you want to open the HTML report? (Y/N)"
if ($viewReport -eq "Y" -or $viewReport -eq "y") {
    Start-Process $reportPath
}

```

---

Revision #2

Created 2025-02-26 12:27:20 UTC by Ryan

Updated 2025-03-13 00:56:28 UTC by Ryan