WINDOWS & POWERSHELL COMMANDS

ESSENTIAL GUIDE FOR CYBERSECURITY PROFESSIONALS

OKAN YILDIZ

JULY 2025

Windows and PowerShell Commands: Essential Guide for Cybersecurity Professionals

Introduction

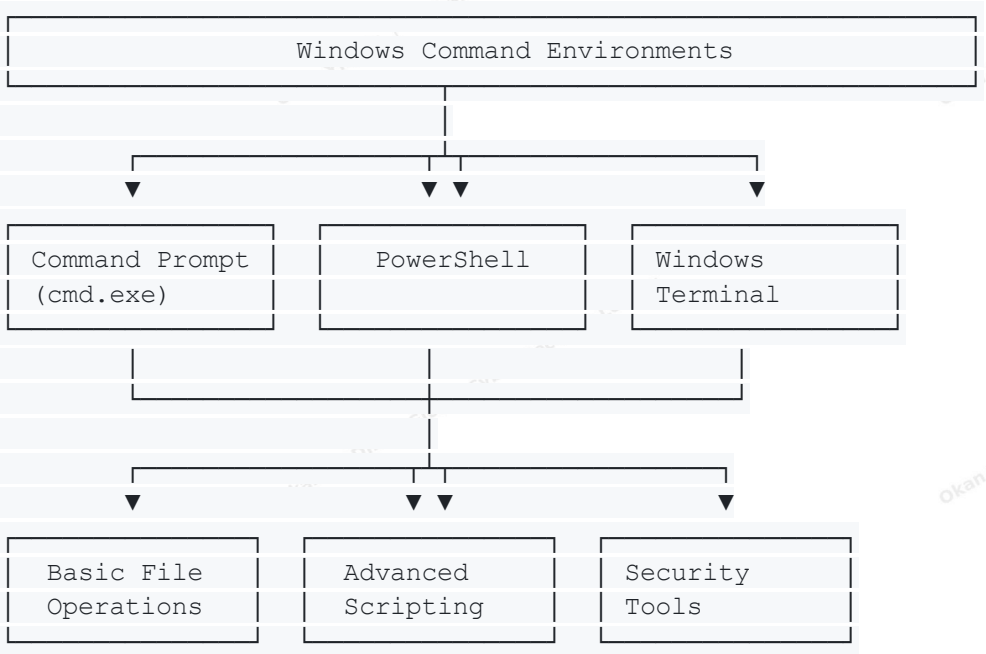
In the realm of cybersecurity, proficiency with command-line interfaces provides security professionals with powerful capabilities for system investigation, threat hunting, incident response, and security automation. Windows environments present unique challenges and opportunities for security analysis, with native command-line tools and PowerShell offering robust mechanisms to detect, analyze, and respond to security incidents. This comprehensive guide explores the most crucial Windows and PowerShell commands leveraged by cybersecurity professionals, providing detailed technical implementation guidance, practical use cases, and security considerations for each.

While graphical tools offer convenience, command-line proficiency enables security professionals to implement repeatable processes, perform complex analyses, and develop automated security workflows that scale across enterprise environments. The commands and techniques covered in this article represent the foundational toolkit that security practitioners use to secure Windows environments, investigate suspicious activities, and respond to security incidents effectively.

Windows Command Line Essentials for Security Professionals

Understanding the Windows Command Environment

Windows offers multiple command-line interfaces, each with distinct capabilities for security operations:



Essential File System Commands for Security Analysis

File system analysis forms the foundation of many security operations:

```
# Find all files modified in the last 24 hours (PowerShell)
Get-ChildItem -Path C:\ -Recurse -Force -ErrorAction SilentlyContinue |
    Where-Object { $_.LastWriteTime -gt (Get-Date).AddDays(-1) } |
    Select-Object FullName, LastWriteTime, Length

# Find hidden files using Command Prompt
dir /a:h C:\Users\username /s
```

Practical Security Applications:

Identifying Recently Modified System Files: Modified system files may indicate compromise or malicious activity. Security professionals regularly audit file modifications on critical system paths and compare against known-good baselines.

```
# Check for modified files in Windows system directory
Get-ChildItem -Path C:\Windows\System32 -Recurse -Force -ErrorAction
SilentlyContinue |
    Where-Object { $_.LastWriteTime -gt (Get-Date).AddDays(-7) } |
    Select-Object FullName, LastWriteTime, Length |
    Sort-Object LastWriteTime -Descending |
    Format-Table -AutoSize
```

File Integrity Verification: Computing and comparing file hashes helps verify file integrity and identify potentially malicious modifications.

```
# Generate and compare file hashes
Get-FileHash -Path C:\Windows\System32\drivers\etc\hosts -Algorithm SHA256
Get-FileHash -Path C:\path\to\baseline\hosts -Algorithm SHA256
```

The Command Prompt equivalent uses the `certutil` utility:

```
certutil -hashfile C:\Windows\System32\drivers\etc\hosts SHA256
```

Network Analysis Commands

Network analysis commands help security professionals identify suspicious connections, potentially compromised systems, and data exfiltration channels:

```
# Display all active network connections and listening ports
netstat -ano

# Display detailed network connection statistics with process names (PowerShell)
Get-NetTCPConnection |
    Select-Object LocalAddress, LocalPort, RemoteAddress, RemotePort, State,
OwningProcess,
        @{Name="ProcessName";Expression={(Get-Process -Id
$_.OwningProcess).Name}} |
    Sort-Object ProcessName
```

Advanced Security Applications:

Identifying Suspicious Network Connections: Regular monitoring of network connections helps identify command-and-control (C2) channels and unauthorized communications.

```
# Find all connections to non-standard ports
Get-NetTCPConnection -State Established |
    Where-Object { $_.RemotePort -notin @(80, 443, 53, 25, 587, 143, 993) } |
    ForEach-Object {
        $process = Get-Process -Id $_.OwningProcess -ErrorAction
SilentlyContinue
        [PSCustomObject]@{
            LocalAddress = $_.LocalAddress
            LocalPort = $_.LocalPort
            RemoteAddress = $_.RemoteAddress
            RemotePort = $_.RemotePort
            State = $_.State
            ProcessName = $process.Name
            ProcessPath = $process.Path
            CreationTime = $process.StartTime
        }
    } | Format-Table -AutoSize
```

DNS Cache Analysis: Examining DNS cache entries helps identify potentially malicious domain resolutions.

```
# Display DNS cache entries
Get-DnsClientCache | Select-Object Name, Data, TimeToLive
```

Command Prompt equivalent:

```
# Display DNS cache entries
Get-DnsClientCache | Select-Object Name, Data, TimeToLive
```

Process Management for Security Operations

Understanding running processes is essential for identifying malicious software and unauthorized activities:

```
# List all running processes with detailed information
Get-Process | Select-Object Name, Id, Path, Company, CPU, WorkingSet, StartTime
|
    Sort-Object WorkingSet -Descending | Format-Table -AutoSize

# List all running services
Get-Service | Where-Object {$_.Status -eq "Running"} |
    Select-Object Name, DisplayName, Status
```

Security Analysis Techniques:

Identifying Suspicious Processes: Unusual process names, locations, or parent-child relationships can indicate malicious activity.

```
# Find processes without a valid path (often suspicious)
Get-Process | Where-Object { $_.Path -eq $null -and $_.Name -ne "Idle" -and
$_.Name -ne "System" } |
    Select-Object Name, Id, SessionId
```

Analyzing Process Arguments: Examining command-line arguments can reveal suspicious behavior.

```
# Get command line parameters for all processes
Get-WmiObject Win32_Process | Select-Object ProcessId, Name, CommandLine |
    Sort-Object Name | Format-Table -AutoSize
```

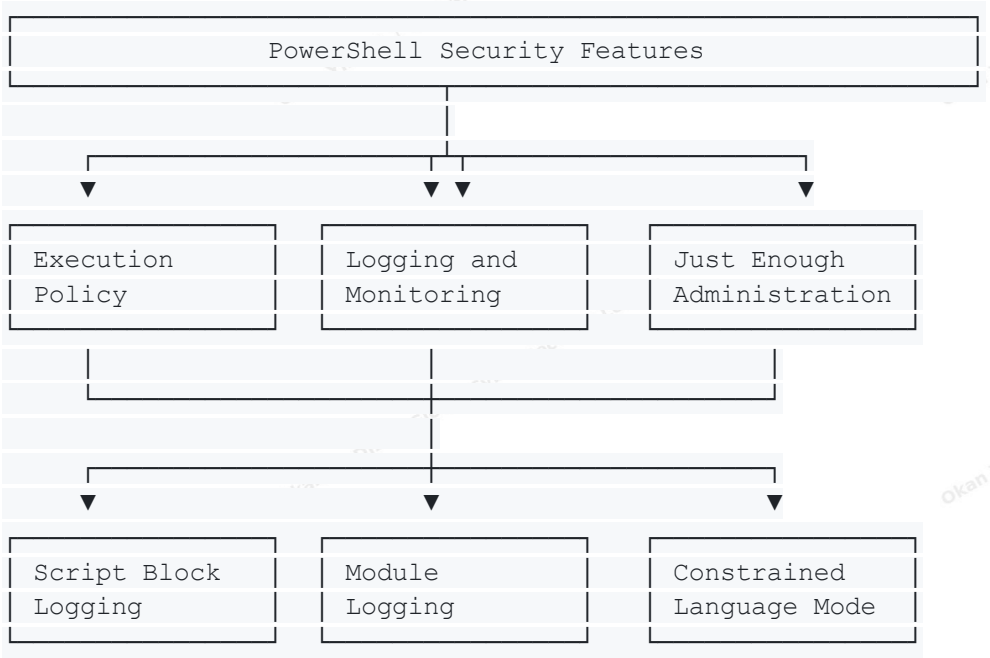
Using WMI through Command Prompt:

```
wmic process get processid,name,commandline
```

PowerShell for Security Analysis and Incident Response

PowerShell Security Features and Considerations

PowerShell's security capabilities make it both a powerful security tool and a potential attack vector:



PowerShell Execution Policy: Controls which scripts can run on the system.

```
# Get current execution policy
Get-ExecutionPolicy -List

# Set execution policy to restricted for security
Set-ExecutionPolicy -ExecutionPolicy Restricted -Scope LocalMachine
```

Enhanced Logging and Monitoring: PowerShell logs provide invaluable data for security analysis.

```
# Enable enhanced PowerShell logging
$settings = @{
    EnableScriptBlockLogging = $true
    EnableTranscription = $true
    EnableModuleLogging = $true
    ModuleNames = @('*')
    TranscriptionSettings = @{
        EnableInvocationHeader = $true
        OutputDirectory = 'C:\PowerShellLogs'
    }
}

# This would typically be set in Group Policy
foreach ($key in $settings.Keys) {
    Write-Host "Configure: $key = $($settings[$key])"
}
```

PowerShell Event Log Analysis for Security

PowerShell's event logs contain crucial security information:

```
# Retrieve PowerShell script block logging events
Get-WinEvent -LogName "Microsoft-Windows-PowerShell/Operational" |
    Where-Object { $_.Id -eq 4104 } |
    Select-Object TimeCreated, Message -First 20

# Find potentially malicious encoded commands
Get-WinEvent -LogName "Microsoft-Windows-PowerShell/Operational" |
    Where-Object { $_.Message -like "*encodedcommand*" -or $_.Message -like
"*-enc*" } |
    Select-Object TimeCreated, Id, Message
```

Advanced Security Analysis:

Detecting PowerShell-Based Attacks: Analyze PowerShell logs for indicators of malicious activity.

```
# Search for common PowerShell attack patterns
$maliciousPatterns = @(
    'Invoke-Expression',
    'IEX',
    'Invoke-Mimikatz',
    'Invoke-Obfuscation',
    'Net.WebClient',
    'DownloadString',
    'DownloadFile',
    'EncodedCommand',
    'Hidden Window',
    'New-Object',
    '-Enc',
    'FromBase64String'
)

$events = Get-WinEvent -LogName "Microsoft-Windows-PowerShell/Operational"
-MaxEvents 5000 -ErrorAction SilentlyContinue

$suspiciousEvents = $events | Where-Object {
    $message = $_.Message
    $foundPatterns = $maliciousPatterns | Where-Object { $message -match $_ }
    $foundPatterns -ne $null
} | Select-Object TimeCreated, Id, @{Name="MatchedPatterns";Expression={
    ($maliciousPatterns | Where-Object { $_.Message -match $_ }) -join ', '
}}, Message

$suspiciousEvents | Format-Table -AutoSize
```

PowerShell Use-After-Free Analysis: Identify suspicious PowerShell instances that might be running.

```
# Find processes where PowerShell is running
Get-WmiObject Win32_Process -Filter "Name='powershell.exe'" |
    Select-Object ProcessId, Name, CommandLine,

@{Name="CreationDate";Expression={$_.ConvertToDateTime($_.Creation
Date)}} |
    Format-Table -AutoSize
```

Security Automation with PowerShell

PowerShell enables security professionals to automate routine security tasks:

```
# Automated security baseline checker function
function Test-SecurityBaseline {
    param (
        [Parameter(Mandatory=$true)]
        [string]$ComputerName
    )

    $results = @{
        ComputerName = $ComputerName
        FirewallEnabled = $false
        UpdateStatus = "Unknown"
        AntivirusStatus = "Unknown"
        SuspiciousServices = @()
        PasswordPolicy = @{}
        Vulnerabilities = @()
    }

    # Check Windows Firewall status
    try {
        $firewallProfiles = Invoke-Command -ComputerName $ComputerName
-ScriptBlock {
            Get-NetFirewallProfile
        }
        $results.FirewallEnabled = -not ($firewallProfiles | Where-Object {
$_.Enabled -eq $false })
    } catch {
        $results.FirewallEnabled = "Error: $_"
    }

    # Check for Windows updates
    try {
        $updateSession = Invoke-Command -ComputerName $ComputerName -ScriptBlock
{
            New-Object -ComObject Microsoft.Update.Session
            $updater = $updateSession.CreateUpdateSearcher()
            $updater.Search("IsInstalled=0")
        }
        $results.UpdateStatus = "$($updateSession.Updates.Count) updates
pending"
    } catch {
        $results.UpdateStatus = "Error checking updates: $_"
    }

    # Add more security checks here...

    return [PSCustomObject]$results
}
```

```
# Example usage
Test-SecurityBaseline -ComputerName "localhost"
```

Windows Security Commands and Utilities

Built-in Windows Security Tools

Windows provides several native security tools accessible via command line:

Windows Defender from Command Line:

```
# Run a quick scan with Windows Defender
Start-MpScan -ScanType QuickScan

# Get Windows Defender status
Get-MpComputerStatus | Select-Object AMServiceEnabled, AntispywareEnabled,
AntivirusEnabled, BehaviorMonitorEnabled, IoavProtectionEnabled,
RealTimeProtectionEnabled

# Update Windows Defender signatures
Update-MpSignature
```

System File Checker for Integrity Verification:

```
# Verify system file integrity
sfc /scannow

# Verify and repair Windows system image
DISM /Online /Cleanup-Image /RestoreHealth
```

Windows Event Log Analysis:

```
# Search for failed login attempts
Get-WinEvent -LogName Security |
    Where-Object { $_.Id -eq 4625 } |
    Select-Object TimeCreated,
        @{Name="Username";Expression={$_.Properties[5].Value}},

@{Name="SourceComputer";Expression={$_.Properties[13].Value}},
        @{Name="Status";Expression={$_.Properties[7].Value}},

@{Name="FailureReason";Expression={$_.Properties[8].Value}} |
```

```
    Format-Table -AutoSize

# Search for account lockouts
Get-WinEvent -LogName Security |
    Where-Object { $_.Id -eq 4740 } |
    Select-Object TimeCreated,

@{Name="LockedAccount";Expression={$_.Properties[0].Value}} |
    Format-Table -AutoSize
```

Security Configuration Analysis

Analyzing system security configurations helps identify potential vulnerabilities:

```
# Get local security policy settings
secedit /export /cfg C:\secpol.cfg
Get-Content C:\secpol.cfg

# Get Group Policy settings report
gpresult /h C:\GPReport.html

# Analyze system services startup configuration
Get-Service | Select-Object Name, DisplayName, StartType, Status |
    Where-Object { $_.StartType -eq "Automatic" } |
    Sort-Object Name |
    Format-Table -AutoSize
```

Practical Security Applications:

User Account Analysis: Identifying user accounts with elevated privileges or unusual settings.

```
# Get local user accounts and their properties
Get-LocalUser | Select-Object Name, Enabled, PasswordRequired,
    PasswordLastSet, LastLogon, AccountExpires |
    Format-Table -AutoSize

# Find local administrators
Get-LocalGroupMember -Group "Administrators" |
    Select-Object Name, PrincipalSource, ObjectClass
```

Share Permission Analysis: Identifying overly permissive network shares.

```
# Get shared folders and permissions
$shares = Get-WmiObject -Class Win32_Share
```

```
foreach ($share in $shares) {
    $acl = Get-Acl -Path $share.Path -ErrorAction SilentlyContinue
    if ($acl) {
        [PSCustomObject]@{
            ShareName = $share.Name
            Path = $share.Path
            Description = $share.Description
            Permissions = ($acl.Access | Select-Object
IdentityReference, FileSystemRights)
        }
    }
}
```

Registry Analysis for Security

The Windows Registry contains crucial security-related settings and potential indicators of compromise:

```
# Check for autostart programs (common persistence mechanism)
Get-ItemProperty -Path 'HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\Run' |
    Format-Table -AutoSize

# Check for suspicious browser extensions (Chrome example)
Get-ChildItem -Path 'HKCU:\Software\Google\Chrome\Extensions' -ErrorAction
SilentlyContinue |
    ForEach-Object {
        Get-ItemProperty -Path "Registry::$_" | Select-Object -Property name,
version, path
    }
```

Advanced Registry Analysis:

Detecting Persistence Mechanisms: Malware often uses registry for persistence.

```
# Common Registry persistence locations
$persistenceLocations = @(
    'HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\Run',
    'HKCU:\SOFTWARE\Microsoft\Windows\CurrentVersion\Run',
    'HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnce',
    'HKCU:\SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnce',
    'HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\RunServices',
    'HKCU:\SOFTWARE\Microsoft\Windows\CurrentVersion\RunServices',
    'HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\RunServicesOnce',
    'HKCU:\SOFTWARE\Microsoft\Windows\CurrentVersion\RunServicesOnce',
    'HKLM:\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon\Userinit',
    'HKLM:\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon\Shell',
```

```
    'HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\Explorer\Run',
    'HKCU:\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\Explorer\Run'
)

foreach ($location in $persistenceLocations) {
    if (Test-Path -Path $location) {
        Write-Host "Checking $location" -ForegroundColor Cyan
        Get-ItemProperty -Path $location |
            Format-Table -AutoSize
    }
}
```

Registry Forensic Analysis: Finding evidence of recently executed programs or accessed files.

```
# Get recently opened files from registry
Get-ItemProperty -Path
'HKCU:\Software\Microsoft\Windows\CurrentVersion\Explorer\RecentDocs\.exe'
-ErrorAction SilentlyContinue |
    Select-Object -ExpandProperty MRUListEx |
    ForEach-Object {
        if ($_ -ne 0) {
            try {
                [System.Text.Encoding]::Unicode.GetString((Get-ItemProperty
-Path
'HKCU:\Software\Microsoft\Windows\CurrentVersion\Explorer\RecentDocs\.exe')."$_"
)
            } catch {}
        }
    }
```

Advanced PowerShell Techniques for Security Operations

Remote Security Assessment with PowerShell

PowerShell's remoting capabilities enable security professionals to perform remote assessments:

```
# Enable secure PowerShell remoting
Enable-PSRemoting -Force
Set-Item WSMan:\localhost\Client\TrustedHosts -Value "RemoteComputer" -Force

# Perform remote security assessment
Invoke-Command -ComputerName RemoteComputer -ScriptBlock {
```

```
    # Get security event log information
    Get-EventLog -LogName Security -Newest 10 |
        Select-Object TimeGenerated, EntryType, Source, EventID, Message

    # Get running processes
    Get-Process | Select-Object Name, Id, Path

    # Check firewall status
    Get-NetFirewallProfile | Select-Object Name, Enabled
} -Credential (Get-Credential)
```

Security Considerations:

Secure Authentication: Always use encrypted connections and strong authentication.

```
# Configure HTTPS for PowerShell remoting
# First, create and install a certificate
$params = @{
    CertStoreLocation = "Cert:\LocalMachine\My"
    DnsName = $env:COMPUTERNAME
    FriendlyName = "PowerShell Remoting Certificate"
    NotAfter = (Get-Date).AddYears(1)
}
$cert = New-SelfSignedCertificate @params

# Configure WinRM to use HTTPS transport with the certificate
New-Item -Path WSMan:\LocalHost\Listener -Transport HTTPS -Address *
-CertificateThumbprint $cert.Thumbprint -Force
```

Just Enough Administration (JEA): Limit remote PowerShell capabilities to only what's needed.

```
# Create a JEA configuration file
New-PSSessionConfigurationFile -Path C:\JEA\SecurityAudit.pssc `
    -SessionType RestrictedRemoteServer `
    -VisibleCmdlets Get-Process, Get-Service, Get-EventLog `
    -VisibleFunctions 'Get-DiskSpace', 'Get-FirewallStatus' `
    -LanguageMode NoLanguage

# Register the JEA endpoint
Register-PSSessionConfiguration -Path C:\JEA\SecurityAudit.pssc `
    -Name SecurityAudit `
    -ShowSecurityDescriptorUI
```

Threat Hunting with PowerShell

PowerShell enables effective threat hunting across Windows environments:

```
# Hunt for unusual PowerShell usage
$suspiciousPowerShellUsage = Get-WinEvent -LogName
"Microsoft-Windows-PowerShell/Operational" |
    Where-Object { $_.Message -match "-enc" -or $_.Message -match
"downloadstring" -or $_.Message -match "hidden" } |
    Select-Object TimeCreated, Id, Message

# Hunt for LOLBAS (Living Off The Land Binaries And Scripts) usage
$lolbasBinaries = @(
    "certutil.exe",
    "regsvr32.exe",
    "mshta.exe",
    "bitsadmin.exe",
    "regasm.exe",
    "regsvcs.exe",
    "msbuild.exe",
    "installutil.exe",
    "rundll32.exe",
    "odbcconf.exe",
    "wmic.exe"
)

$lolbasEvents = Get-WinEvent -LogName "Security" -MaxEvents 10000 |
    Where-Object {
        $_.Id -eq 4688 -and
        ($lolbasBinaries | ForEach-Object { $_.Message -match $_ })
    } |
    Select-Object TimeCreated, Id,
        @{Name="CommandLine";Expression={$_.Properties[8].Value}},
        @{Name="User";Expression={$_.Properties[1].Value}}
```

Advanced Hunting Techniques:

File Hash-Based Threat Hunting: Use file hashes to identify potentially malicious files.

```
# Get file hashes of all executable files in a directory and compare against
known bad
$knownBadHashes = @(
    "e5d7d48f991a345351c47397ed67e74001f9a0fb3f8bc337b094f41757a945f4",
    "54e6ea47eb04634d3e87fd7787e2136ccfbcc80ade06f5bdb3d390065a1f5435"
)

Get-ChildItem -Path "C:\Program Files" -Include "*.exe", "*.dll" -Recurse |
    ForEach-Object {
```

```
        try {
            $hash = Get-FileHash -Path $_.FullName -Algorithm SHA256
-ErrorAction Stop
            if ($knownBadHashes -contains $hash.Hash) {
                [PSCustomObject]@{
                    FilePath = $_.FullName
                    FileHash = $hash.Hash
                    MatchedKnownBad = $true
                }
            }
        } catch {}
    }
```

Memory Analysis for Indicators of Compromise: Examine process memory for suspicious patterns.

```
# Load memory analysis module
Import-Module .\MemoryTools.psm1

# Scan process memory for indicators of compromise (conceptual example)
$iocPatterns = @(
    "PsExec",
    "mimikatz",
    "Invoke-Mimikatz",
    "Invoke-Expression",
    "FromBase64String",
    "hidden window",
    "net user /add",
    "powershell -enc"
)

foreach ($process in (Get-Process)) {
    foreach ($pattern in $iocPatterns) {
        $result = Search-ProcessMemory -Id $process.Id -Pattern $pattern
-ErrorAction SilentlyContinue
        if ($result) {
            [PSCustomObject]@{
                ProcessId = $process.Id
                ProcessName = $process.Name
                MatchedPattern = $pattern
                MemoryRegion = $result.Address
            }
        }
    }
}
```

Incident Response with PowerShell

PowerShell provides essential capabilities for incident response activities:

```
# Incident response script to collect volatile data
function Invoke-IncidentResponse {
    param (
        [Parameter(Mandatory=$true)]
        [string]$OutputPath
    )

    # Create output directory
    $timestamp = Get-Date -Format "yyyyMMdd-HHmmss"
    $casePath = Join-Path -Path $OutputPath -ChildPath "IR_$timestamp"
    New-Item -Path $casePath -ItemType Directory -Force | Out-Null

    # Collect system information
    $computerInfo = Get-ComputerInfo
    $computerInfo | ConvertTo-Json -Depth 3 | Out-File -FilePath
"$casePath\SystemInfo.json"

    # Collect running processes
    $processes = Get-Process | Select-Object Name, Id, Path, Company, Product,
        StartTime, CPU, WorkingSet, SessionId,
        @{Name="MemoryUsageMB";Expression={[math]::Round($_.WorkingSet / 1MB,
2)}}
    $processes | Export-Csv -Path "$casePath\Processes.csv" -NoTypeInformation

    # Collect network connections
    $connections = Get-NetTCPConnection |
        Select-Object LocalAddress, LocalPort, RemoteAddress, RemotePort, State,
OwningProcess,
            @{Name="ProcessName";Expression={(Get-Process -Id $_.OwningProcess
-ErrorAction SilentlyContinue).Name}}
    $connections | Export-Csv -Path "$casePath\NetworkConnections.csv"
-NoTypeInformation

    # Collect DNS cache
    $dnsCache = Get-DnsClientCache
    $dnsCache | Export-Csv -Path "$casePath\DNSCache.csv" -NoTypeInformation

    # Collect loaded drivers
    $drivers = Get-WmiObject Win32_SystemDriver |
        Select-Object Name, Description, PathName, ServiceType, StartMode, State
    $drivers | Export-Csv -Path "$casePath\Drivers.csv" -NoTypeInformation

    # Collect scheduled tasks
    $tasks = Get-ScheduledTask |
        Select-Object TaskName, TaskPath, State,
            @{Name="Actions";Expression={$_.Actions.Execute}},
```

```
            @{Name="Arguments";Expression={$_.Actions.Arguments}}
    $tasks | Export-Csv -Path "$casePath\ScheduledTasks.csv" -NoTypeInformation

    # Collect service information
    $services = Get-Service |
        Select-Object Name, DisplayName, Status, StartType,
            @{Name="PathName";Expression={
                (Get-WmiObject -Class Win32_Service -Filter
"Name='$($_.Name)'").PathName
            }}
    $services | Export-Csv -Path "$casePath\Services.csv" -NoTypeInformation

    # Collect recent event logs
    $securityLogs = Get-WinEvent -LogName Security -MaxEvents 1000 -ErrorAction
SilentlyContinue
    $securityLogs | Export-Clixml -Path "$casePath\SecurityLogs.xml"

    $systemLogs = Get-WinEvent -LogName System -MaxEvents 1000 -ErrorAction
SilentlyContinue
    $systemLogs | Export-Clixml -Path "$casePath\SystemLogs.xml"

    $appLogs = Get-WinEvent -LogName Application -MaxEvents 1000 -ErrorAction
SilentlyContinue
    $appLogs | Export-Clixml -Path "$casePath\ApplicationLogs.xml"

    $powershellLogs = Get-WinEvent -LogName
"Microsoft-Windows-PowerShell/Operational" -MaxEvents 1000 -ErrorAction
SilentlyContinue
    $powershellLogs | Export-Clixml -Path "$casePath\PowerShellLogs.xml"

    # Collect autorun information
    $autorunLocations = @(
        "HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\Run",
        "HKCU:\SOFTWARE\Microsoft\Windows\CurrentVersion\Run",
        "HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnce",
        "HKCU:\SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnce"
    )

    $autoruns = foreach ($location in $autorunLocations) {
        if (Test-Path -Path $location) {
            Get-ItemProperty -Path $location |
                Get-Member -MemberType NoteProperty |
                Where-Object { $_.Name -notin @('PSPath', 'PSParentPath',
'PSChildName', 'PSDrive', 'PSProvider') } |
                ForEach-Object {
                    [PSCustomObject]@{
                        Location = $location
                        Name = $_.Name
                        Value = (Get-ItemProperty -Path $location -Name
$_.Name).$($_.Name)
                    }
```

```
                }
            }
        }
        $autoruns | Export-Csv -Path "$casePath\Autoruns.csv" -NoTypeInformation

        # Create a summary report
        $report = @"
# Incident Response Collection Summary
Collection Date: $(Get-Date)
Computer Name: $($computerInfo.CsName)
OS Version: $($computerInfo.WindowsProductName) $($computerInfo.WindowsVersion)

## Collection Statistics
- Processes: $($processes.Count)
- Network Connections: $($connections.Count)
- Services: $($services.Count)
- Scheduled Tasks: $($tasks.Count)
- Autoruns: $($autoruns.Count)
- Security Log Events: $($securityLogs.Count)
- System Log Events: $($systemLogs.Count)
- Application Log Events: $($appLogs.Count)
- PowerShell Log Events: $($powershellLogs.Count)

## Collection Path
$casePath
"@

        $report | Out-File -FilePath "$casePath\CollectionSummary.md"

        return $casePath
}

# Example usage
$collectionPath = Invoke-IncidentResponse -OutputPath "C:\IR_Collections"
```

Windows Defender and Security Management

Windows Defender from PowerShell

Windows Defender can be fully controlled through PowerShell for advanced security operations:

```
# Get Windows Defender preferences
Get-MpPreference | Format-List *enabled*

# Get threat detection history
Get-MpThreatDetection | Select-Object ThreatID, Resources, InitialDetectionTime,
```

```
ProcessName, DetectionSource

# Remove specific threat by ID
Remove-MpThreat -ThreatID '2147519003'

# Configure Defender exclusions
Add-MpPreference -ExclusionPath "C:\Safe\Directory"
Add-MpPreference -ExclusionProcess "TrustedApp.exe"
```

Advanced Windows Defender Management:

Configuring Advanced Threat Protection:

```
# Enable cloud-delivered protection
Set-MpPreference -MAPSReporting Advanced

# Enable network protection
Set-MpPreference -EnableNetworkProtection Enabled

# Enable controlled folder access (ransomware protection)
Set-MpPreference -EnableControlledFolderAccess Enabled

# Add protected folders
Add-MpPreference -ControlledFolderAccessProtectedFolders "C:\Important"

# Add allowed applications for controlled folder access
Add-MpPreference -ControlledFolderAccessAllowedApplications "C:\Program
Files\App\app.exe"
```

Scheduled Scanning Configuration:

```
# Configure scheduled scan settings
Set-MpPreference -ScanScheduleDay Everyday
Set-MpPreference -ScanScheduleTime 02:00
Set-MpPreference -ScanParameters FullScan
```

Security Policy Management

Managing security policies via command line enables consistent security configurations:

```
# Export current security policy
secedit /export /cfg C:\temp\secpol.cfg

# View account policies
$secpol = Get-Content C:\temp\secpol.cfg | Out-String
```

```
$passwordPolicy = $secpol -match
"PasswordComplexity|MinimumPasswordLength|PasswordHistorySize|MaximumPasswordAge
"
$passwordPolicy

# Modify security policy (example for minimum password length)
(Get-Content C:\temp\secpol.cfg) -replace 'MinimumPasswordLength = .*',
'MinimumPasswordLength = 14' |
    Set-Content C:\temp\secpol.cfg

# Apply modified security policy
secedit /configure /db C:\temp\secdb.sdb /cfg C:\temp\secpol.cfg /areas
SECURITYPOLICY
```

Advanced Command Line Forensics and Analysis

Memory Analysis with PowerShell

Memory analysis provides crucial forensic evidence during incident response:

```
# Create a memory dump of a specific process
$process = Get-Process -Name "suspicious_process"
$dumpFile = "$env:TEMP\memdump_$($process.Id).dmp"

Add-Type -TypeDefinition @"
using System;
using System.Runtime.InteropServices;

public class ProcessDump {
    [DllImport("dbghelp.dll", SetLastError = true)]
    public static extern bool MiniDumpWriteDump(
        IntPtr hProcess,
        uint ProcessId,
        IntPtr hFile,
        uint DumpType,
        IntPtr ExceptionParam,
        IntPtr UserStreamParam,
        IntPtr CallbackParam);
}
"@

$fs = New-Object System.IO.FileStream($dumpFile, [System.IO.FileMode]::Create)
$result = [ProcessDump]::MiniDumpWriteDump(
    $process.Handle,
    $process.Id,
    $fs.SafeFileHandle.DangerousGetHandle(),
    2, # MiniDumpWithFullMemory
```

```
    [IntPtr]::Zero,
    [IntPtr]::Zero,
    [IntPtr]::Zero
)
$fs.Close()

if ($result) {
    Write-Host "Memory dump created successfully: $dumpFile"
} else {
    Write-Host "Failed to create memory dump. Error code:
$([System.Runtime.InteropServices.Marshal]::GetLastWin32Error())"
}
```

Advanced Event Log Analysis

Event logs contain valuable evidence for security investigations:

```
# Define a function to convert Windows Security Event IDs to meaningful
descriptions
function Get-EventIdDescription {
    param (
        [Parameter(Mandatory=$true)]
        [int]$EventId
    )

    $descriptions = @{
        4624 = "Successful logon"
        4625 = "Failed logon attempt"
        4634 = "Account logoff"
        4648 = "Logon attempt with explicit credentials"
        4657 = "Registry value modified"
        4663 = "Object access attempt"
        4688 = "Process creation"
        4698 = "Scheduled task created"
        4699 = "Scheduled task deleted"
        4700 = "Scheduled task enabled"
        4701 = "Scheduled task disabled"
        4702 = "Scheduled task updated"
        4720 = "User account created"
        4722 = "User account enabled"
        4724 = "Password reset attempt"
        4728 = "Member added to security-enabled global group"
        4732 = "Member added to security-enabled local group"
        4738 = "User account changed"
        4740 = "User account locked out"
        4756 = "Member added to security-enabled universal group"
        4767 = "User account unlocked"
        4771 = "Kerberos pre-authentication failed"
```

```
        4776 = "The domain controller attempted to validate the credentials for
an account"
        4778 = "Session reconnected to window station"
        4779 = "Session disconnected from window station"
        4798 = "A user's local group membership was enumerated"
        4799 = "A security-enabled local group membership was enumerated"
        5140 = "Network share was accessed"
        5145 = "Network share object was checked to see if client can be granted
desired access"
        7045 = "New service was installed"
    }

    if ($descriptions.ContainsKey($EventId)) {
        return $descriptions[$EventId]
    } else {
        return "Unknown event ID"
    }
}

# Advanced function to search for suspicious authentication events
function Find-SuspiciousAuthenticationEvents {
    param (
        [Parameter()]
        [int]$Hours = 24,

        [Parameter()]
        [string]$ExportPath
    )

    $startTime = (Get-Date).AddHours(-$Hours)
    Write-Host "Searching for suspicious authentication events since $startTime"

    # Get failed logon attempts
    $failedLogons = Get-WinEvent -FilterHashtable @{
        LogName = 'Security'
        Id = 4625
        StartTime = $startTime
    } -ErrorAction SilentlyContinue |
    Select-Object TimeCreated, Id,
        @{Name="EventDescription";Expression={Get-EventIdDescription -EventId
$_.Id}},
        @{Name="Username";Expression={$_.Properties[5].Value}},
        @{Name="Domain";Expression={$_.Properties[6].Value}},
        @{Name="SourceWorkstation";Expression={$_.Properties[13].Value}},
        @{Name="SourceIP";Expression={$_.Properties[19].Value}},
        @{Name="LogonType";Expression={$_.Properties[10].Value}},
        @{Name="Status";Expression={$_.Properties[7].Value}},
        @{Name="SubStatus";Expression={$_.Properties[9].Value}}

    # Get successful logons
    $successfulLogons = Get-WinEvent -FilterHashtable @{
```

```
        LogName = 'Security'
        Id = 4624
        StartTime = $startTime
    } -ErrorAction SilentlyContinue |
    Select-Object TimeCreated, Id,
        @{Name="EventDescription";Expression={Get-EventIdDescription -EventId
$_.Id}},
        @{Name="Username";Expression={$_.Properties[5].Value}},
        @{Name="Domain";Expression={$_.Properties[6].Value}},
        @{Name="SourceWorkstation";Expression={$_.Properties[11].Value}},
        @{Name="SourceIP";Expression={$_.Properties[18].Value}},
        @{Name="LogonType";Expression={$_.Properties[8].Value}}

    # Get account lockouts
    $accountLockouts = Get-WinEvent -FilterHashtable @{
        LogName = 'Security'
        Id = 4740
        StartTime = $startTime
    } -ErrorAction SilentlyContinue |
    Select-Object TimeCreated, Id,
        @{Name="EventDescription";Expression={Get-EventIdDescription -EventId
$_.Id}},
        @{Name="Username";Expression={$_.Properties[0].Value}},
        @{Name="SourceComputer";Expression={$_.Properties[1].Value}}

    # Analyze for patterns
    $suspiciousActivities = @()

    # Group failed logons by username
    $failedLogonsByUser = $failedLogons | Group-Object -Property Username

    foreach ($userGroup in $failedLogonsByUser) {
        if ($userGroup.Count -ge 5) {
            # High number of failed logons for one user
            $suspiciousActivities += [PSCustomObject]@{
                Type = "Multiple Failed Logons"
                Username = $userGroup.Name
                Count = $userGroup.Count
                TimeRange = "$($userGroup.Group[0].TimeCreated) to
$($userGroup.Group[-1].TimeCreated)"
                SourceIPs = ($userGroup.Group.SourceIP | Sort-Object -Unique)
-join ", "
                Details = "Multiple failed logon attempts for single user"
            }
        }
    }

    # Look for successful logon after failures
    foreach ($userGroup in $failedLogonsByUser) {
        $username = $userGroup.Name
        $lastFailedTime = ($userGroup.Group | Sort-Object TimeCreated
```

```
-Descending)[0].TimeCreated

        $successAfterFailure = $successfulLogons |
            Where-Object { $_.Username -eq $username -and $_.TimeCreated -gt
$lastFailedTime } |
            Sort-Object TimeCreated |
            Select-Object -First 1

        if ($successAfterFailure) {
            $timeDiff = New-TimeSpan -Start $lastFailedTime -End
$successAfterFailure.TimeCreated

            if ($timeDiff.TotalMinutes -lt 30) {
                $suspiciousActivities += [PSCustomObject]@{
                    Type = "Success After Failures"
                    Username = $username
                    FailedCount = $userGroup.Count
                    SuccessTime = $successAfterFailure.TimeCreated
                    TimeSinceLastFailure =
"$([math]::Round($timeDiff.TotalMinutes, 2)) minutes"
                    SourceIP = $successAfterFailure.SourceIP
                    Details = "Successful logon shortly after multiple failures"
                }
            }
        }
    }

    # Look for successful logons from unusual sources
    $usualSourcesByUser = @{}
    foreach ($logon in $successfulLogons) {
        $key = "$($logon.Username)_$($logon.Domain)"
        if (-not $usualSourcesByUser.ContainsKey($key)) {
            $usualSourcesByUser[$key] = @()
        }
        $usualSourcesByUser[$key] += $logon.SourceIP
    }

    foreach ($key in $usualSourcesByUser.Keys) {
        $sources = $usualSourcesByUser[$key]
        $uniqueSources = $sources | Sort-Object -Unique

        if ($uniqueSources.Count -gt 3) {
            $username, $domain = $key.Split('_')
            $suspiciousActivities += [PSCustomObject]@{
                Type = "Multiple Logon Sources"
                Username = $username
                Domain = $domain
                SourceCount = $uniqueSources.Count
                Sources = $uniqueSources -join ", "
                Details = "User logged in from multiple different source IPs"
            }
```

```
        }
    }

    # Look for after-hours logons
    $afterHoursLogons = $successfulLogons | Where-Object {
        $hour = $_.TimeCreated.Hour
        $hour -lt 6 -or $hour -gt 18
    }

    if ($afterHoursLogons) {
        $suspiciousActivities += [PSCustomObject]@{
            Type = "After Hours Logon"
            Count = $afterHoursLogons.Count
            Users = ($afterHoursLogons.Username | Sort-Object -Unique) -join ",
"
            Details = "Logons detected outside normal business hours (6 AM - 6
PM)"
        }
    }

    # Output results
    Write-Host "Analysis Results:"
    Write-Host "-----------------"
    Write-Host "Failed Logons: $($failedLogons.Count)"
    Write-Host "Successful Logons: $($successfulLogons.Count)"
    Write-Host "Account Lockouts: $($accountLockouts.Count)"
    Write-Host "Suspicious Activities: $($suspiciousActivities.Count)"

    # Export results if path provided
    if ($ExportPath) {
        $failedLogons | Export-Csv -Path "$ExportPath\FailedLogons.csv"
-NoTypeInformation
        $successfulLogons | Export-Csv -Path "$ExportPath\SuccessfulLogons.csv"
-NoTypeInformation
        $accountLockouts | Export-Csv -Path "$ExportPath\AccountLockouts.csv"
-NoTypeInformation
        $suspiciousActivities | Export-Csv -Path
"$ExportPath\SuspiciousActivities.csv" -NoTypeInformation

        Write-Host "Results exported to $ExportPath"
    }

    return $suspiciousActivities
}

# Example usage
$suspiciousEvents = Find-SuspiciousAuthenticationEvents -Hours 48 -ExportPath
"C:\SecurityAudit"
$suspiciousEvents | Format-Table -AutoSize
```

Integrating Windows and PowerShell Security Commands

Building Security Automation Workflows

Combining PowerShell and Windows commands enables comprehensive security workflows:

```
# Comprehensive security assessment script
function Invoke-SecurityAssessment {
    param (
        [Parameter(Mandatory=$true)]
        [string]$ComputerName,

        [Parameter()]
        [string]$OutputPath = ".\SecurityAssessment",

        [Parameter()]
        [switch]$IncludeEventLogs,

        [Parameter()]
        [switch]$IncludeRegistry,

        [Parameter()]
        [int]$EventLogDays = 3
    )

    $timestamp = Get-Date -Format "yyyyMMdd-HHmmss"
    $assessmentPath = Join-Path -Path $OutputPath -ChildPath
"${ComputerName}_${timestamp}"
    New-Item -Path $assessmentPath -ItemType Directory -Force | Out-Null

    Write-Host "Starting security assessment for $ComputerName" -ForegroundColor
Cyan
    Write-Host "Results will be saved to $assessmentPath" -ForegroundColor Cyan

    try {
        # System Information
        Write-Host "Collecting system information..." -ForegroundColor Green
        $systemInfo = Invoke-Command -ComputerName $ComputerName -ScriptBlock {
            Get-ComputerInfo | Select-Object CsName, CsDomain, CsManufacturer,
CsModel,
                OsName, OsVersion, OsBuildNumber, OsArchitecture,
                WindowsProductName, WindowsVersion
        }
        $systemInfo | ConvertTo-Json | Out-File -FilePath
"$assessmentPath\SystemInfo.json"
```

```
        # Windows Updates
        Write-Host "Checking Windows Update status..." -ForegroundColor Green
        $updates = Invoke-Command -ComputerName $ComputerName -ScriptBlock {
            Get-HotFix | Sort-Object InstalledOn -Descending |
                Select-Object HotFixID, Description, InstalledOn
        }
        $updates | Export-Csv -Path "$assessmentPath\InstalledUpdates.csv"
-NoTypeInformation

        # Security Configuration
        Write-Host "Analyzing security configuration..." -ForegroundColor Green
        $securityConfig = Invoke-Command -ComputerName $ComputerName
-ScriptBlock {
            # Check Windows Defender status
            $defenderStatus = Get-MpComputerStatus | Select-Object
AMServiceEnabled, AntivirusEnabled,
                RealTimeProtectionEnabled, IoavProtectionEnabled, NISEnabled

            # Check firewall status
            $firewallStatus = Get-NetFirewallProfile | Select-Object Name,
Enabled

            # Check BitLocker status
            $bitlockerStatus = Get-BitLockerVolume -ErrorAction SilentlyContinue
|
                Select-Object MountPoint, VolumeStatus, EncryptionPercentage,
ProtectionStatus

            # Check UAC configuration
            $uacStatus = Get-ItemProperty -Path
"HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\System" -Name
"EnableLUA", "ConsentPromptBehaviorAdmin" -ErrorAction SilentlyContinue

            # Return composite object
            [PSCustomObject]@{
                WindowsDefender = $defenderStatus
                Firewall = $firewallStatus
                BitLocker = $bitlockerStatus
                UAC = $uacStatus
            }
        }
        $securityConfig | ConvertTo-Json -Depth 4 | Out-File -FilePath
"$assessmentPath\SecurityConfiguration.json"

        # User Account Analysis
        Write-Host "Analyzing user accounts..." -ForegroundColor Green
        $accounts = Invoke-Command -ComputerName $ComputerName -ScriptBlock {
            # Get local user accounts
            $localUsers = Get-LocalUser | Select-Object Name, Enabled,
PasswordRequired,
                PasswordLastSet, LastLogon, AccountExpires, Description
```

```
            # Get local administrators
            $admins = Get-LocalGroupMember -Group "Administrators" |
                Select-Object Name, PrincipalSource, ObjectClass

            # Return composite object
            [PSCustomObject]@{
                LocalUsers = $localUsers
                Administrators = $admins
            }
        }
        $accounts.LocalUsers | Export-Csv -Path "$assessmentPath\LocalUsers.csv"
-NoTypeInformation
        $accounts.Administrators | Export-Csv -Path
"$assessmentPath\LocalAdministrators.csv" -NoTypeInformation

        # Network Configuration
        Write-Host "Analyzing network configuration..." -ForegroundColor Green
        $networkConfig = Invoke-Command -ComputerName $ComputerName -ScriptBlock
{
            # Get network adapters
            $adapters = Get-NetAdapter | Select-Object Name,
InterfaceDescription, Status, MacAddress,
                LinkSpeed, MediaType, PhysicalMediaType

            # Get IP configuration
            $ipConfig = Get-NetIPConfiguration | Select-Object InterfaceAlias,
InterfaceIndex,
                IPv4Address, IPv4DefaultGateway, DNSServer

            # Get listening ports
            $netstat = Get-NetTCPConnection -State Listen |
                Select-Object LocalAddress, LocalPort, State,
                    @{Name="ProcessId";Expression={$_.OwningProcess}},
                    @{Name="ProcessName";Expression={(Get-Process -Id
$_.OwningProcess -ErrorAction SilentlyContinue).Name}}

            # Get SMB shares
            $shares = Get-SmbShare | Select-Object Name, Path, Description

            # Return composite object
            [PSCustomObject]@{
                NetworkAdapters = $adapters
                IPConfiguration = $ipConfig
                ListeningPorts = $netstat
                Shares = $shares
            }
        }
        $networkConfig.NetworkAdapters | Export-Csv -Path
"$assessmentPath\NetworkAdapters.csv" -NoTypeInformation
        $networkConfig.IPConfiguration | Export-Csv -Path
```

```
"$assessmentPath\IPConfiguration.csv" -NoTypeInformation
        $networkConfig.ListeningPorts | Export-Csv -Path
"$assessmentPath\ListeningPorts.csv" -NoTypeInformation
        $networkConfig.Shares | Export-Csv -Path
"$assessmentPath\NetworkShares.csv" -NoTypeInformation

        # Startup Items
        Write-Host "Analyzing startup configuration..." -ForegroundColor Green
        $startupItems = Invoke-Command -ComputerName $ComputerName -ScriptBlock
{
            # Get scheduled tasks
            $tasks = Get-ScheduledTask | Where-Object { $_.State -ne "Disabled"
} |
                Select-Object TaskName, TaskPath, State,
                    @{Name="Actions";Expression={$_.Actions.Execute}},
                    @{Name="Arguments";Expression={$_.Actions.Arguments}}

            # Get startup programs from registry
            $startupPrograms = @()
            $startupLocations = @(
                "HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\Run",
                "HKCU:\SOFTWARE\Microsoft\Windows\CurrentVersion\Run",
                "HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnce",
                "HKCU:\SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnce"
            )

            foreach ($location in $startupLocations) {
                if (Test-Path -Path $location) {
                    $items = Get-ItemProperty -Path $location
                    foreach ($prop in ($items | Get-Member -MemberType
NoteProperty).Name) {
                        if ($prop -notmatch '^PS') {
                            $startupPrograms += [PSCustomObject]@{
                                Name = $prop
                                Command = $items.$prop
                                Location = $location
                            }
                        }
                    }
                }
            }

            # Return composite object
            [PSCustomObject]@{
                ScheduledTasks = $tasks
                StartupPrograms = $startupPrograms
            }
        }
        $startupItems.ScheduledTasks | Export-Csv -Path
"$assessmentPath\ScheduledTasks.csv" -NoTypeInformation
        $startupItems.StartupPrograms | Export-Csv -Path
```

```
"$assessmentPath\StartupPrograms.csv" -NoTypeInformation

        # Services Analysis
        Write-Host "Analyzing services..." -ForegroundColor Green
        $services = Invoke-Command -ComputerName $ComputerName -ScriptBlock {
            Get-Service | Select-Object Name, DisplayName, Status, StartType,
                @{Name="PathName";Expression={
                    (Get-WmiObject -Class Win32_Service -Filter
"Name='$($_.Name)'").PathName
                }},
                @{Name="Account";Expression={
                    (Get-WmiObject -Class Win32_Service -Filter
"Name='$($_.Name)'").StartName
                }}
        }
        $services | Export-Csv -Path "$assessmentPath\Services.csv"
-NoTypeInformation

        # Process analysis
        Write-Host "Analyzing running processes..." -ForegroundColor Green
        $processes = Invoke-Command -ComputerName $ComputerName -ScriptBlock {
            Get-Process | Select-Object Name, Id, Path, Company, Product,
                StartTime, CPU, WorkingSet, SessionId,
                @{Name="MemoryUsageMB";Expression={[math]::Round($_.WorkingSet /
1MB, 2)}}
        }
        $processes | Export-Csv -Path "$assessmentPath\RunningProcesses.csv"
-NoTypeInformation

        # Windows Defender status and scan results
        Write-Host "Checking Windows Defender status..." -ForegroundColor Green
        $defenderStatus = Invoke-Command -ComputerName $ComputerName
-ScriptBlock {
            # Get Windows Defender preferences
            $prefs = Get-MpPreference | Select-Object *enabled*, *paths*,
*mode*, *setting*

            # Get protection status
            $status = Get-MpComputerStatus | Select-Object AntivirusEnabled,
RealTimeProtectionEnabled,
                IoavProtectionEnabled, AntispywareEnabled,
BehaviorMonitorEnabled,
                AntivirusSignatureLastUpdated, AMServiceEnabled, AMRunningMode

            # Get threat history
            $threats = Get-MpThreatDetection | Select-Object ThreatID,
ThreatName,
                SeverityID, CategoryID, InitialDetectionTime, ProcessName,
Resources

            # Return composite object
```

```
            [PSCustomObject]@{
                Preferences = $prefs
                Status = $status
                ThreatHistory = $threats
            }
        }
        $defenderStatus.Status | ConvertTo-Json | Out-File -FilePath
"$assessmentPath\DefenderStatus.json"
        $defenderStatus.Preferences | ConvertTo-Json | Out-File -FilePath
"$assessmentPath\DefenderPreferences.json"
        $defenderStatus.ThreatHistory | Export-Csv -Path
"$assessmentPath\DefenderThreatHistory.csv" -NoTypeInformation

        # Event Log Analysis (if requested)
        if ($IncludeEventLogs) {
            Write-Host "Analyzing event logs (this may take some time)..."
-ForegroundColor Green
            $startDate = (Get-Date).AddDays(-$EventLogDays)

            # Security log analysis
            $securityEvents = Invoke-Command -ComputerName $ComputerName
-ScriptBlock {
                param($startDate)

                # Get failed logon attempts
                $failedLogons = Get-WinEvent -FilterHashtable @{
                    LogName = 'Security'
                    Id = 4625
                    StartTime = $startDate
                } -ErrorAction SilentlyContinue |
                Select-Object TimeCreated, Id,
                    @{Name="Username";Expression={$_.Properties[5].Value}},
                    @{Name="Domain";Expression={$_.Properties[6].Value}},

@{Name="SourceWorkstation";Expression={$_.Properties[13].Value}},
                    @{Name="SourceIP";Expression={$_.Properties[19].Value}},
                    @{Name="Status";Expression={$_.Properties[7].Value}},
                    @{Name="SubStatus";Expression={$_.Properties[9].Value}}

                # Get account lockouts
                $accountLockouts = Get-WinEvent -FilterHashtable @{
                    LogName = 'Security'
                    Id = 4740
                    StartTime = $startDate
                } -ErrorAction SilentlyContinue |
                Select-Object TimeCreated, Id,
                    @{Name="Username";Expression={$_.Properties[0].Value}}

                # Get privilege usage
                $privilegeUsage = Get-WinEvent -FilterHashtable @{
                    LogName = 'Security'
```

```
                    Id = 4672
                    StartTime = $startDate
                } -ErrorAction SilentlyContinue -MaxEvents 100 |
                Select-Object TimeCreated, Id,
                    @{Name="Username";Expression={$_.Properties[1].Value}},
                    @{Name="PrivilegeList";Expression={$_.Properties[4].Value}}

                # Get account modifications
                $accountMods = Get-WinEvent -FilterHashtable @{
                    LogName = 'Security'
                    Id = @(4720, 4722, 4724, 4738)
                    StartTime = $startDate
                } -ErrorAction SilentlyContinue |
                Select-Object TimeCreated, Id,
                    @{Name="EventType";Expression={
                        switch ($_.Id) {
                            4720 { "Account Created" }
                            4722 { "Account Enabled" }
                            4724 { "Password Reset" }
                            4738 { "Account Modified" }
                        }
                    }},
                    @{Name="TargetAccount";Expression={$_.Properties[0].Value}},
                    @{Name="PerformedBy";Expression={$_.Properties[4].Value}}

                # Return composite object
                [PSCustomObject]@{
                    FailedLogons = $failedLogons
                    AccountLockouts = $accountLockouts
                    PrivilegeUsage = $privilegeUsage
                    AccountModifications = $accountMods
                }
            } -ArgumentList $startDate

            # Create event logs directory
            $eventLogPath = Join-Path -Path $assessmentPath -ChildPath
"EventLogs"
            New-Item -Path $eventLogPath -ItemType Directory -Force | Out-Null

            $securityEvents.FailedLogons | Export-Csv -Path
"$eventLogPath\FailedLogons.csv" -NoTypeInformation
            $securityEvents.AccountLockouts | Export-Csv -Path
"$eventLogPath\AccountLockouts.csv" -NoTypeInformation
            $securityEvents.PrivilegeUsage | Export-Csv -Path
"$eventLogPath\PrivilegeUsage.csv" -NoTypeInformation
            $securityEvents.AccountModifications | Export-Csv -Path
"$eventLogPath\AccountModifications.csv" -NoTypeInformation
        }

        # Registry analysis (if requested)
        if ($IncludeRegistry) {
```

```
            Write-Host "Analyzing registry for security configurations..."
-ForegroundColor Green
            $registrySettings = Invoke-Command -ComputerName $ComputerName
-ScriptBlock {
                # Security policy settings
                $securityPolicy = @{}

                # Password policy
                try {
                    $passwordPolicy = Get-ItemProperty -Path
"HKLM:\SYSTEM\CurrentControlSet\Services\Netlogon\Parameters" -ErrorAction
SilentlyContinue
                    $securityPolicy["PasswordPolicy"] = $passwordPolicy
                } catch {}

                # Account lockout policy
                try {
                    $lockoutPolicy = Get-ItemProperty -Path
"HKLM:\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon" -ErrorAction
SilentlyContinue
                    $securityPolicy["LockoutPolicy"] = $lockoutPolicy
                } catch {}

                # Audit policy
                try {
                    $auditPolicy = Get-ItemProperty -Path
"HKLM:\SYSTEM\CurrentControlSet\Control\Lsa" -ErrorAction SilentlyContinue
                    $securityPolicy["AuditPolicy"] = $auditPolicy
                } catch {}

                # Remote desktop settings
                try {
                    $rdpSettings = Get-ItemProperty -Path
"HKLM:\SYSTEM\CurrentControlSet\Control\Terminal Server" -ErrorAction
SilentlyContinue
                    $securityPolicy["RDPSettings"] = $rdpSettings
                } catch {}

                # SMB settings
                try {
                    $smbSettings = Get-ItemProperty -Path
"HKLM:\SYSTEM\CurrentControlSet\Services\LanmanServer\Parameters" -ErrorAction
SilentlyContinue
                    $securityPolicy["SMBSettings"] = $smbSettings
                } catch {}

                # Return security policy settings
                return $securityPolicy
            }

            $registrySettings | ConvertTo-Json -Depth 3 | Out-File -FilePath
```

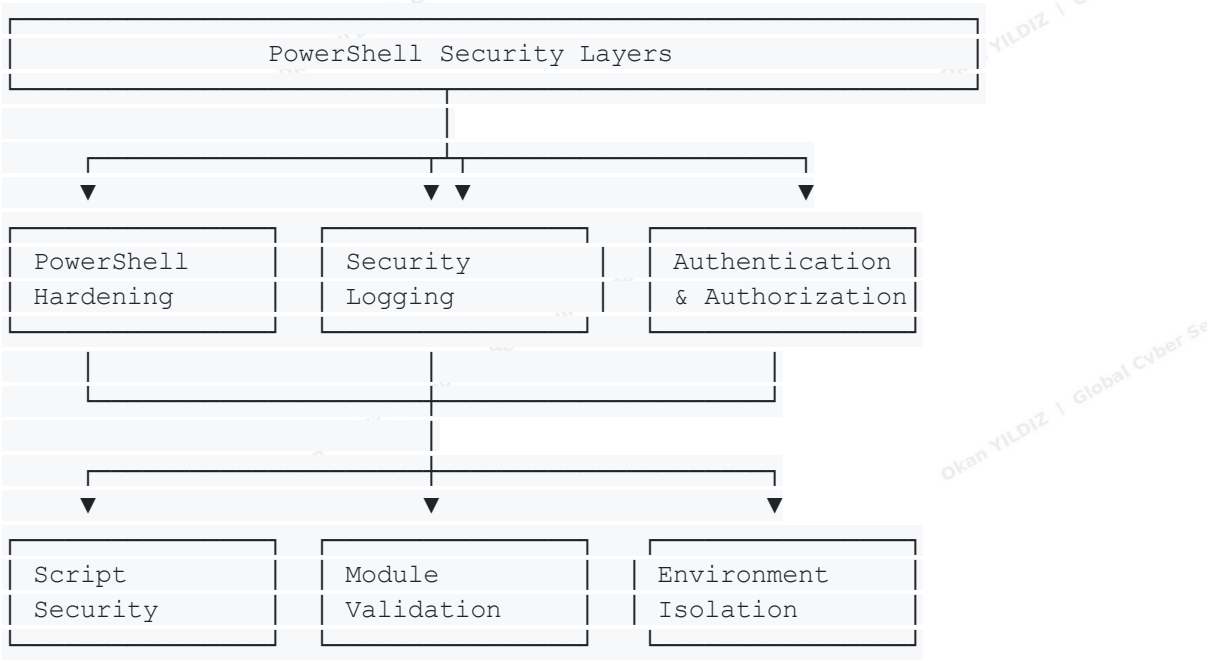
```
"$assessmentPath\RegistrySecuritySettings.json"
        }

        # Generate executive summary
        Write-Host "Generating security assessment summary..." -ForegroundColor
Green
        $summaryReport = @"
# Security Assessment Report
**Computer Name:** $($systemInfo.CsName)
**Date:** $(Get-Date -Format "yyyy-MM-dd HH:mm:ss")

## System Information
- **Computer Name:** $($systemInfo.CsName)
- **Domain:** $($systemInfo.CsDomain)
- **OS:** $($systemInfo.WindowsProductName)
- **Version:** $($systemInfo.WindowsVersion)
- **Architecture:** $($systemInfo.OsArchitecture)

## Security Configuration Status
- **Windows Defender Enabled:**
$($securityConfig.WindowsDefender.AntivirusEnabled)
- **Real-Time Protection:**
$($securityConfig.WindowsDefender.RealTimeProtectionEnabled)
- **Firewall Enabled (Domain):** $($securityConfig.Firewall | Where-Object Name
-eq 'Domain' | Select-Object -ExpandProperty Enabled)
- **BitLocker Status:** $(if ($securityConfig.BitLocker) { "Enabled on
$($securityConfig.BitLocker.Count) volumes" } else { "Not enabled" })
- **UAC Enabled:** $($securityConfig.UAC.EnableLUA -eq 1)

## User Account Status
- **Local User Accounts:** $($accounts.LocalUsers.Count)
- **Enabled Local Accounts:** $($accounts.LocalUsers | Where-Object Enabled -eq
$true | Measure-Object).Count
- **Local Administrators:** $($accounts.Administrators.Count)

## Network Configuration
- **Network Adapters:** $($networkConfig.NetworkAdapters.Count)
- **Active Network Adapters:** $($networkConfig.NetworkAdapters | Where-Object
Status -eq 'Up' | Measure-Object).Count
- **Open Network Shares:** $($networkConfig.Shares.Count)
- **Listening Ports:** $($networkConfig.ListeningPorts.Count)

## Services and Processes
- **Running Services:** $($services | Where-Object Status -eq 'Running' |
Measure-Object).Count
- **Automatic Startup Services:** $($services | Where-Object StartType -eq
'Automatic' | Measure-Object).Count
- **Running Processes:** $($processes.Count)

## Update Status
- **Installed Updates:** $($updates.Count)
```

```
- **Last Update:** $($updates | Select-Object -First 1 -ExpandProperty
InstalledOn)

## Startup Configuration
- **Scheduled Tasks:** $($startupItems.ScheduledTasks.Count)
- **Startup Programs:** $($startupItems.StartupPrograms.Count)

## Windows Defender Status
- **Antivirus Enabled:** $($defenderStatus.Status.AntivirusEnabled)
- **Signature Last Updated:**
$($defenderStatus.Status.AntivirusSignatureLastUpdated)
- **Detected Threats:** $($defenderStatus.ThreatHistory.Count)

## Security Assessment Results
The following areas require attention:

1. $(if (-not $securityConfig.WindowsDefender.RealTimeProtectionEnabled) {
"Windows Defender real-time protection is disabled" } else { "Windows Defender
is properly configured" })
2. $(if (($securityConfig.Firewall | Where-Object { -not $_.Enabled }).Count -gt
0) { "One or more firewall profiles are disabled" } else { "All firewall
profiles are enabled" })
3. $(if (-not $securityConfig.BitLocker) { "BitLocker is not enabled on any
drives" } else { "BitLocker is properly configured" })
4. $(if ($accounts.LocalUsers | Where-Object { $_.Enabled -and
$_.PasswordLastSet -lt (Get-Date).AddDays(-90) }) { "One or more local user
accounts have passwords older than 90 days" } else { "Local user account
passwords are within policy" })
5. $(if ($networkConfig.Shares.Count -gt 3) { "Multiple network shares are
configured which may increase attack surface" } else { "Network share
configuration appears reasonable" })
6. $(if ($defenderStatus.Status.AntivirusSignatureLastUpdated -lt
(Get-Date).AddDays(-7)) { "Windows Defender signatures are outdated" } else {
"Windows Defender signatures are up to date" })

## Recommended Actions
1. $(if (-not $securityConfig.WindowsDefender.RealTimeProtectionEnabled) {
"Enable Windows Defender real-time protection" } else { "Continue monitoring
Windows Defender status" })
2. $(if (($securityConfig.Firewall | Where-Object { -not $_.Enabled }).Count -gt
0) { "Enable all Windows Firewall profiles" } else { "Continue monitoring
firewall configuration" })
3. $(if (-not $securityConfig.BitLocker) { "Enable BitLocker on all system
volumes" } else { "Verify BitLocker recovery keys are safely stored" })
4. $(if ($accounts.LocalUsers | Where-Object { $_.Enabled -and
$_.PasswordLastSet -lt (Get-Date).AddDays(-90) }) { "Reset passwords for
accounts with passwords older than 90 days" } else { "Continue enforcing
password policies" })
5. Review all network shares and remove unnecessary shares
6. $(if ($defenderStatus.Status.AntivirusSignatureLastUpdated -lt
(Get-Date).AddDays(-7)) { "Update Windows Defender signatures immediately" }
```

```
else { "Ensure Windows Defender signatures continue to update automatically" })
7. Review startup programs and scheduled tasks for potentially unwanted
applications
8. Ensure system and application updates are applied regularly

"@
        $summaryReport | Out-File -FilePath
"$assessmentPath\SecurityAssessmentSummary.md"

        Write-Host "Security assessment completed successfully!"
-ForegroundColor Green
        Write-Host "Results saved to $assessmentPath" -ForegroundColor Green

        return $assessmentPath
    }
    catch {
        Write-Host "Error during security assessment: $_" -ForegroundColor Red
        Write-Host "Stack Trace: $($_.ScriptStackTrace)" -ForegroundColor Red
        throw $_
    }
}

# Example usage
$assessmentPath = Invoke-SecurityAssessment -ComputerName "localhost"
-IncludeEventLogs -IncludeRegistry
```

Securing PowerShell Environments

PowerShell Security Best Practices



To harden PowerShell environments while preserving its security capabilities:

```
# PowerShell Security Hardening Script
function Enable-PowerShellSecurity {
    param (
        [Parameter()]
        [switch]$Enforce,

        [Parameter()]
        [switch]$JustShowCommands
    )

    $securitySettings = @(
        @{
            Name = "Enable PowerShell Script Block Logging"
            Command = 'Set-ItemProperty -Path
"HKLM:\SOFTWARE\Policies\Microsoft\Windows\PowerShell\ScriptBlockLogging" -Name
"EnableScriptBlockLogging" -Value 1 -Type DWORD -Force'
            Description = "Records the content of all script blocks that
PowerShell processes, providing visibility into potentially malicious scripts"
        },
        @{
            Name = "Enable PowerShell Module Logging"
            Command = 'Set-ItemProperty -Path
"HKLM:\SOFTWARE\Policies\Microsoft\Windows\PowerShell\ModuleLogging" -Name
"EnableModuleLogging" -Value 1 -Type DWORD -Force; Set-ItemProperty -Path
"HKLM:\SOFTWARE\Policies\Microsoft\Windows\PowerShell\ModuleLogging\ModuleNames"
-Name "*" -Value "*" -Type String -Force'
            Description = "Logs the pipeline execution details of PowerShell
modules, helping detect malicious module usage"
        },
        @{
            Name = "Enable PowerShell Transcription"
            Command = 'Set-ItemProperty -Path
"HKLM:\SOFTWARE\Policies\Microsoft\Windows\PowerShell\Transcription" -Name
"EnableTranscription" -Value 1 -Type DWORD -Force; Set-ItemProperty -Path
"HKLM:\SOFTWARE\Policies\Microsoft\Windows\PowerShell\Transcription" -Name
"EnableInvocationHeader" -Value 1 -Type DWORD -Force; Set-ItemProperty -Path
"HKLM:\SOFTWARE\Policies\Microsoft\Windows\PowerShell\Transcription" -Name
"OutputDirectory" -Value "C:\PowerShellLogs" -Type String -Force'
            Description = "Creates text-based transcripts of PowerShell
sessions, capturing input and output for forensic analysis"
        },
        @{
            Name = "Set PowerShell Execution Policy to RemoteSigned"
            Command = 'Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope
LocalMachine -Force'
            Description = "Requires that all scripts downloaded from the
internet be signed by a trusted publisher before execution"
```

```
        },
        @{
            Name = "Enable Windows Remote Management Access Control"
            Command = 'Set-PSSessionConfiguration -Name Microsoft.PowerShell
-showSecurityDescriptorUI'
            Description = "Configures access control for PowerShell remoting to
limit who can connect"
        },
        @{
            Name = "Enable PowerShell Protected Event Logging"
            Command = 'Set-ItemProperty -Path
"HKLM:\SOFTWARE\Policies\Microsoft\Windows\EventLog\ProtectedEventLogging" -Name
"EnableProtectedEventLogging" -Value 1 -Type DWORD -Force'
            Description = "Enables protected event logging which prevents
modification of event logs"
        },
        @{
            Name = "Configure WSMAN MaxMemoryPerShellMB limit"
            Command = 'Set-Item WSMan:\localhost\Shell\MaxMemoryPerShellMB 1024'
            Description = "Limits the amount of memory that can be used by a
remote PowerShell session, mitigating resource exhaustion attacks"
        },
        @{
            Name = "Enable Extended Protection for Authentication in WinRM"
            Command = 'Set-Item WSMan:\localhost\Service\Auth\ExtendedProtection
-Value "Always"'
            Description = "Enhances the security of authentication by preventing
authentication relay attacks"
        },
        @{
            Name = "Create PowerShell Log Directory"
            Command = 'New-Item -Path "C:\PowerShellLogs" -ItemType Directory
-Force; $acl = Get-Acl -Path "C:\PowerShellLogs"; $accessRule = New-Object
System.Security.AccessControl.FileSystemAccessRule("SYSTEM", "FullControl",
"ContainerInherit,ObjectInherit", "None", "Allow");
$acl.SetAccessRule($accessRule); $accessRule = New-Object
System.Security.AccessControl.FileSystemAccessRule("Administrators",
"FullControl", "ContainerInherit,ObjectInherit", "None", "Allow");
$acl.SetAccessRule($accessRule); Set-Acl -Path "C:\PowerShellLogs" -AclObject
$acl'
            Description = "Creates a secure directory for PowerShell transcripts
with appropriate permissions"
        }
    )

    if ($JustShowCommands) {
        foreach ($setting in $securitySettings) {
            Write-Host "Setting: $($setting.Name)" -ForegroundColor Cyan
            Write-Host "Description: $($setting.Description)" -ForegroundColor
Yellow
            Write-Host "Command: $($setting.Command)" -ForegroundColor Green
```

```
            Write-Host "-----------------------------------"
        }
        return
    }

    foreach ($setting in $securitySettings) {
        Write-Host "Configuring: $($setting.Name)" -ForegroundColor Cyan
        Write-Host "  $($setting.Description)" -ForegroundColor Yellow

        if ($Enforce) {
            try {
                # Create required registry paths if they don't exist
                if ($setting.Command -match
"HKLM:\\SOFTWARE\\Policies\\Microsoft\\Windows\\PowerShell") {
                    # Ensure the PowerShell policy registry path exists
                    if (-not (Test-Path
"HKLM:\SOFTWARE\Policies\Microsoft\Windows\PowerShell")) {
                        New-Item -Path
"HKLM:\SOFTWARE\Policies\Microsoft\Windows\PowerShell" -Force | Out-Null
                    }

                    # Create specific paths
                    if ($setting.Command -match "ScriptBlockLogging" -and -not
(Test-Path
"HKLM:\SOFTWARE\Policies\Microsoft\Windows\PowerShell\ScriptBlockLogging")) {
                        New-Item -Path
"HKLM:\SOFTWARE\Policies\Microsoft\Windows\PowerShell\ScriptBlockLogging" -Force
| Out-Null
                    }

                    if ($setting.Command -match "ModuleLogging" -and -not
(Test-Path
"HKLM:\SOFTWARE\Policies\Microsoft\Windows\PowerShell\ModuleLogging")) {
                        New-Item -Path
"HKLM:\SOFTWARE\Policies\Microsoft\Windows\PowerShell\ModuleLogging" -Force |
Out-Null
                        New-Item -Path
"HKLM:\SOFTWARE\Policies\Microsoft\Windows\PowerShell\ModuleLogging\ModuleNames"
-Force | Out-Null
                    }

                    if ($setting.Command -match "Transcription" -and -not
(Test-Path
"HKLM:\SOFTWARE\Policies\Microsoft\Windows\PowerShell\Transcription")) {
                        New-Item -Path
"HKLM:\SOFTWARE\Policies\Microsoft\Windows\PowerShell\Transcription" -Force |
Out-Null
                    }
                }

                # Execute the command
```

```
                $scriptBlock = [ScriptBlock]::Create($setting.Command)
                Invoke-Command -ScriptBlock $scriptBlock
                Write-Host "  Successfully applied setting" -ForegroundColor
Green
            }
            catch {
                Write-Host "  Failed to apply setting: $_" -ForegroundColor Red
            }
        }
        else {
            Write-Host "  Run with -Enforce to apply this setting"
-ForegroundColor Yellow
        }
    }

    if ($Enforce) {
        Write-Host "PowerShell security settings have been applied."
-ForegroundColor Green
        Write-Host "Note: Some settings may require a restart of PowerShell to
take effect." -ForegroundColor Yellow
    }
    else {
        Write-Host "This was a dry run. Use -Enforce to apply settings."
-ForegroundColor Yellow
    }
}

# Example usage
Enable-PowerShellSecurity -JustShowCommands
# Enable-PowerShellSecurity -Enforce # Uncomment to actually apply settings
```

Frequently Asked Questions

What are the most important Windows command-line tools for security analysis?

Windows includes several critical command-line tools that security professionals should master:

1. Windows Event Viewer commands (`wevtutil`, PowerShell `Get-WinEvent`): These commands provide access to Windows event logs, which contain valuable information about system and security events including login attempts, process executions, and system changes.
2. PowerShell security modules (`Get-MpComputerStatus`, `Get-Process`, `Get-NetTCPConnection`): PowerShell's security-focused cmdlets provide

detailed information about system state, running processes, and network connections.
3. System integrity verification tools (`sfc`, `DISM`): These tools help verify and repair Windows system files, which is useful for identifying potential tampering.
4. Security configuration tools (`secedit`, `gpresult`): These commands allow security professionals to analyze and export security policy settings.
5. Network diagnostic tools (`netstat`, `ipconfig`, `nslookup`): These provide insights into network configurations and connections, vital for identifying unusual network activity.

The relative importance of these tools varies based on specific security scenarios, but mastering event log analysis through PowerShell's `Get-WinEvent` is perhaps the most universally valuable skill, as it provides access to comprehensive audit data necessary for almost any security investigation.

How can I securely enable PowerShell for security operations while preventing abuse?

Securely enabling PowerShell in an enterprise environment requires a balanced approach:

1. Enable comprehensive logging:
    - Script Block Logging records the content of all script blocks processed
    - Module Logging captures pipeline execution details
    - Transcription creates text-based records of PowerShell sessions
2. These logging mechanisms provide visibility into legitimate and malicious PowerShell usage.
3. Implement Just Enough Administration (JEA):
    - Create limited PowerShell endpoints that restrict available commands
    - Use Role-Based Access Control to define who can use specific capabilities
    - Implement session configurations that limit PowerShell's scope for different administrators
4. Configure execution policy appropriately:
    - Use `RemoteSigned` or `AllSigned` execution policy
    - Implement code signing requirements for production scripts
    - Consider application whitelisting solutions like AppLocker or Windows Defender Application Control
5. Utilize Constrained Language Mode:
    - Limit PowerShell to a subset of language features
    - Restrict access to certain .NET types and methods
    - Combine with Device Guard to enforce constraints
6. Deploy detection capabilities:

- Implement SIEM rules to detect suspicious PowerShell activity
- Create alerts for encoded commands, script downloads, or execution policy bypasses
- Monitor for attempts to disable PowerShell security features

By combining these controls, organizations can leverage PowerShell's security capabilities while significantly reducing the risk of abuse. The specific configuration should be tailored to your organization's security requirements and risk tolerance.

What are the most effective PowerShell commands for rapid incident response?

During a security incident, time is critical. The most effective PowerShell commands for rapid incident response include:

Process analysis and identification of suspicious processes:

```
Get-Process | Select-Object Name, Id, Path, Company, CPU, StartTime |
Sort-Object CPU -Descending
Get-CimInstance Win32_Process | Select-Object ProcessId, ParentProcessId, Name,
CommandLine, CreationDate
```

Network connection analysis to identify command and control traffic:

```
Get-NetTCPConnection -State Established |
Select-Object LocalAddress, LocalPort, RemoteAddress, RemotePort, State,
OwningProcess,
@{Name="ProcessName";Expression={(Get-Process -Id $_.OwningProcess).Name}} |
Sort-Object ProcessName
```

Event log analysis for suspicious authentication events:

```
Get-WinEvent -FilterHashtable @{LogName='Security'; Id=4624,4625,4634,4647,4648}
-MaxEvents 100 |
Select-Object TimeCreated, Id, @{Name='EventType';Expression={
  switch ($_.Id) {
    4624 {'Logon'}
    4625 {'Failed Logon'}
    4634 {'Logoff'}
    4647 {'User Initiated Logoff'}
    4648 {'Explicit Credential Logon'}
  }
}}, @{Name='UserName';Expression={$_.Properties[5].Value}}
```

Analysis of recently changed files to identify potential malware:

```
Get-ChildItem -Path C:\ -Recurse -Force -ErrorAction SilentlyContinue |
Where-Object { $_.LastWriteTime -gt (Get-Date).AddDays(-1) } |
Select-Object FullName, LastWriteTime, Length |
Sort-Object LastWriteTime -Descending
```

PowerShell script block logging check for malicious scripts:

```
Get-WinEvent -LogName "Microsoft-Windows-PowerShell/Operational" |
Where-Object { $_.Id -eq 4104 -and ($_.Message -like "*Invoke-Mimikatz*" -or
$_.Message -like "*Invoke-Expression*" -or $_.Message -like
"*[Convert]::FromBase64String*") } |
Select-Object TimeCreated, Id, Message
```

The effectiveness of these commands depends on having appropriate logging enabled before an incident occurs. For the most comprehensive incident response, these commands should be part of a broader incident response plan that includes preserving evidence, containment strategies, and root cause analysis.

How do I automate security auditing with PowerShell across multiple systems?

Automating security audits across multiple systems requires a structured approach:

Create a centralized security scanning framework:

```
function Invoke-EnterpriseSecurityScan {
    param (
        [Parameter(Mandatory=$true)]
        [string[]]$ComputerNames,

        [Parameter()]
        [System.Management.Automation.PSCredential]$Credential,

        [Parameter()]
        [string]$OutputPath = ".\SecurityScans",

        [Parameter()]
        [string[]]$ScanModules = @('SystemInfo', 'SecuritySettings',
```

```
'UserAccounts', 'NetworkConfig', 'EventLogs')
    )

    # Create output directory
    $timestamp = Get-Date -Format "yyyyMMdd-HHmmss"
    $scanPath = Join-Path -Path $OutputPath -ChildPath
"EnterpriseScan_$timestamp"
    New-Item -Path $scanPath -ItemType Directory -Force | Out-Null

    # Define script blocks for different scan modules
    $scanScriptBlocks = @{
        'SystemInfo' = {
            Get-ComputerInfo | Select-Object CsName, CsDomain, OsName,
OsVersion, OsBuildNumber
        }

        'SecuritySettings' = {
            $defenderStatus = Get-MpComputerStatus | Select-Object
AMServiceEnabled, AntivirusEnabled, RealTimeProtectionEnabled
            $firewallStatus = Get-NetFirewallProfile | Select-Object Name,
Enabled

            [PSCustomObject]@{
                ComputerName = $env:COMPUTERNAME
                WindowsDefender = $defenderStatus
                Firewall = $firewallStatus
            }
        }

        'UserAccounts' = {
            $localUsers = Get-LocalUser | Select-Object Name, Enabled,
PasswordRequired, PasswordLastSet, LastLogon
            $localAdmins = Get-LocalGroupMember -Group "Administrators" |
Select-Object Name, PrincipalSource

            [PSCustomObject]@{
                ComputerName = $env:COMPUTERNAME
                LocalUsers = $localUsers
                LocalAdmins = $localAdmins
            }
        }

        'NetworkConfig' = {
            $netAdapters = Get-NetAdapter | Select-Object Name, Status,
MacAddress, LinkSpeed
            $netIpConfig = Get-NetIPConfiguration | Select-Object
InterfaceAlias, IPv4Address, IPv4DefaultGateway
            $tcpConnections = Get-NetTCPConnection -State Listen | Select-Object
LocalAddress, LocalPort, State, OwningProcess

            [PSCustomObject]@{
```

```
                ComputerName = $env:COMPUTERNAME
                NetworkAdapters = $netAdapters
                IPConfiguration = $netIpConfig
                ListeningPorts = $tcpConnections
            }
        }

        'EventLogs' = {
            $failedLogons = Get-WinEvent -FilterHashtable @{LogName='Security';
Id=4625} -MaxEvents 100 -ErrorAction SilentlyContinue |
                Select-Object TimeCreated, Id,
@{Name="Username";Expression={$_.Properties[5].Value}}

            $accountChanges = Get-WinEvent -FilterHashtable
@{LogName='Security'; Id=@(4720,4722,4724,4738)} -MaxEvents 100 -ErrorAction
SilentlyContinue |
                Select-Object TimeCreated, Id,
@{Name="TargetAccount";Expression={$_.Properties[0].Value}}

            [PSCustomObject]@{
                ComputerName = $env:COMPUTERNAME
                FailedLogons = $failedLogons
                AccountChanges = $accountChanges
            }
        }
    }

    # Run scans on each computer
    $results = @{}
    foreach ($computer in $ComputerNames) {
        Write-Host "Scanning $computer..." -ForegroundColor Cyan

        # Create computer-specific results
        $computerResults = @{}
        $computerPath = Join-Path -Path $scanPath -ChildPath $computer
        New-Item -Path $computerPath -ItemType Directory -Force | Out-Null

        # Run each requested scan module
        foreach ($module in $ScanModules) {
            if ($scanScriptBlocks.ContainsKey($module)) {
                Write-Host "  Running $module scan..." -ForegroundColor Yellow

                try {
                    # Use Invoke-Command for remote execution if not local
computer
                    if ($computer -eq $env:COMPUTERNAME -or $computer -eq
"localhost" -or $computer -eq ".") {
                        $moduleResult = Invoke-Command -ScriptBlock
$scanScriptBlocks[$module]
                    }
                    else {
```

```
                        $icmParams = @{
                            ComputerName = $computer
                            ScriptBlock = $scanScriptBlocks[$module]
                        }

                        if ($Credential) {
                            $icmParams.Credential = $Credential
                        }

                        $moduleResult = Invoke-Command @icmParams
                    }

                    # Save module results
                    $computerResults[$module] = $moduleResult

                    # Export module results
                    $moduleResultPath = Join-Path -Path $computerPath -ChildPath
"$module.json"
                    $moduleResult | ConvertTo-Json -Depth 5 | Out-File -FilePath
$moduleResultPath

                    Write-Host "    $module scan completed successfully"
-ForegroundColor Green
                }
                catch {
                    Write-Host "    Error running $module scan: $_"
-ForegroundColor Red
                    $computerResults[$module] = "Error: $_"
                }
            }
            else {
                Write-Host "  Unknown scan module: $module" -ForegroundColor Red
            }
        }

        # Save computer results
        $results[$computer] = $computerResults
    }

    # Generate summary report
    $summaryReport = @"
```

Enterprise Security Scan Summary

```
Scan Date: $(Get-Date -Format "yyyy-MM-dd HH:mm:ss") Computers
Scanned: $($ComputerNames.Count) Scan Modules: $($ScanModules
```

```
-join ", ")
```

Scan Results Summary

Computer	Status	Defender	Firewall	Admin Accounts	Failed Logons
"@					

```
  foreach ($computer in $ComputerNames) {
       $computerResults = $results[$computer]

       $defenderStatus = if
($computerResults.SecuritySettings.WindowsDefender.AntivirusEnabled) { "Enabled"
} else { "Disabled" }
       $firewallStatus = if (($computerResults.SecuritySettings.Firewall |
Where-Object { -not $_.Enabled }).Count -eq 0) { "Enabled" } else { "Partial" }
       $adminCount = if ($computerResults.UserAccounts.LocalAdmins) {
$computerResults.UserAccounts.LocalAdmins.Count } else { "Unknown" }
       $failedLogons = if ($computerResults.EventLogs.FailedLogons) {
$computerResults.EventLogs.FailedLogons.Count } else { "Unknown" }

       $scanStatus = if ($computerResults.Count -eq $ScanModules.Count) {
"Complete" } else { "Partial" }

       $summaryReport += @"

| $computer | $scanStatus | $defenderStatus | $firewallStatus | $adminCount |
$failedLogons | "@ }
  # Add recommendations based on scan results
   $summaryReport += @"
```

Security Recommendations

Based on the scan results, the following recommendations are provided:

```
"@
  # Check for systems with Defender disabled
   $defenderDisabled = $ComputerNames | Where-Object { -not
$results[$_].SecuritySettings.WindowsDefender.AntivirusEnabled }
   if ($defenderDisabled.Count -gt 0) {
       $summaryReport += @"
```

1. Enable Windows Defender on the following systems: $($defenderDisabled -join ", ") "@ } # Check for systems with Firewall disabled $firewallDisabled = $ComputerNames | Where-Object { ($results[$*].SecuritySettings.Firewall | Where-Object { -not $*.Enabled }).Count -gt 0 } if ($firewallDisabled.Count -gt 0) { $summaryReport += @"
2. Enable all Firewall profiles on the following systems: $($firewallDisabled -join ", ") "@ } # Check for systems with excessive admin accounts $excessiveAdmins = $ComputerNames | Where-Object { $results[$_].UserAccounts.LocalAdmins.Count -gt 3 } if ($excessiveAdmins.Count -gt 0) { $summaryReport += @"
3. Review administrator accounts on the following systems which have more than 3 local administrators: $($excessiveAdmins -join ", ") "@ } # Check for systems with failed logons $highFailedLogons = $ComputerNames | Where-Object { $results[$_].EventLogs.FailedLogons.Count -gt 10 } if ($highFailedLogons.Count -gt 0) { $summaryReport += @"
4. Investigate failed logon attempts on the following systems which have more than 10 failed logons: $($highFailedLogons -join ", ") "@ } # Add general recommendations $summaryReport += @"

General Recommendations:

- Ensure all systems have current Windows updates applied
- Review inactive user accounts and disable or remove them
- Implement consistent security baselines across all systems
- Configure centralized security logging and monitoring

Review network shares and remove unnecessary access "@

```
 # Save summary report
 $summaryReport | Out-File -FilePath "$scanPath\EnterpriseScanSummary.md"

 Write-Host "Enterprise security scan completed. Results saved to $scanPath"
-ForegroundColor Green

 return $scanPath
}
```

Example usage

```
$computers = @("localhost", "server1", "server2")
Invoke-EnterpriseSecurityScan -ComputerNames $computers
-OutputPath "C:\SecurityAudits"
```

Schedule automated scanning using Task Scheduler:

```
# Create a scheduled task for regular security scans
$action = New-ScheduledTaskAction -Execute 'PowerShell.exe' -Argument
'-NoProfile -ExecutionPolicy Bypass -File "C:\Scripts\SecurityScan.ps1"'
$trigger = New-ScheduledTaskTrigger -Weekly -DaysOfWeek Sunday -At 3AM
$principal = New-ScheduledTaskPrincipal -UserId "SYSTEM" -LogonType
ServiceAccount -RunLevel Highest
$settings = New-ScheduledTaskSettingsSet -StartWhenAvailable -DontStopOnIdleEnd
-AllowStartIfOnBatteries
Register-ScheduledTask -TaskName "WeeklySecurityScan" -Action $action -Trigger
$trigger -Principal $principal -Settings $settings
```

Monitor the security posture over time:
For effective monitoring, create trend analysis scripts that compare security scan results over time to identify changes in security posture. Implement automated alerting for any significant deviations from baseline security settings.

Integrate results with SIEM systems:
Send scan results to central SIEM solutions for correlation with other security data and automated alerting on security policy violations.

For large enterprises, consider enhancing this approach with PowerShell Just Enough Administration (JEA) to provide secure, limited access for running security audits without granting excessive administrative privileges.

What Windows command-line techniques can help detect malicious persistence mechanisms?

Detecting malicious persistence is critical during security investigations. Here are effective command-line techniques to reveal persistence mechanisms:

Registry autorun locations (PowerShell):

```
# Query common persistence registry locations
$autorunLocations = @(
    "HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\Run",
    "HKCU:\SOFTWARE\Microsoft\Windows\CurrentVersion\Run",
    "HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnce",
    "HKCU:\SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnce",
    "HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\RunServices",
    "HKCU:\SOFTWARE\Microsoft\Windows\CurrentVersion\RunServices",
    "HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\RunServicesOnce",
    "HKCU:\SOFTWARE\Microsoft\Windows\CurrentVersion\RunServicesOnce",
    "HKLM:\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon\Userinit",
    "HKLM:\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon\Shell",
```

```
    "HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\User Shell
Folders",
    "HKCU:\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\User Shell
Folders",

"HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\ShellServiceObjects",

"HKCU:\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\ShellServiceObjects",

"HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\ShellServiceObjectDelayLoad",

"HKCU:\SOFTWARE\Microsoft\Windows\CurrentVersion\ShellServiceObjectDelayLoad"
)

foreach ($location in $autorunLocations) {
    if (Test-Path $location) {
        Write-Host "Checking $location" -ForegroundColor Cyan
        Get-ItemProperty -Path $location |
            Select-Object -Property * -ExcludeProperty PSPath, PSParentPath,
PSChildName, PSDrive, PSProvider |
            Format-Table -AutoSize
    }
}
```

Scheduled Task analysis (Command Prompt and PowerShell):

```
REM Command Prompt version
schtasks /query /fo LIST /v
# PowerShell version with suspicious pattern detection
Get-ScheduledTask | Where-Object {$_.State -ne "Disabled"} |
    ForEach-Object {
        $taskInfo = Get-ScheduledTaskInfo -TaskName $_.TaskName -TaskPath
$_.TaskPath -ErrorAction SilentlyContinue
        $actions = $_.Actions | ForEach-Object { "$($_.Execute) $($_.Arguments)"
}

        $suspicious = $false
        $suspiciousPatterns = @('powershell.exe -e', 'cmd.exe /c',
'rundll32.exe', 'regsvr32.exe', 'wscript.exe',
                                'javascript:', 'vbscript:', 'http://',
'https://', 'IEX', 'DownloadString')

        foreach ($pattern in $suspiciousPatterns) {
            if ($actions -match $pattern) {
                $suspicious = $true
                break
            }
        }
```

```
        [PSCustomObject]@{
            TaskName = $_.TaskName
            TaskPath = $_.TaskPath
            State = $_.State
            LastRunTime = $taskInfo.LastRunTime
            NextRunTime = $taskInfo.NextRunTime
            Actions = $actions -join ";"
            Author = $_.Principal.UserId
            CreationDate = $_.Date
            Suspicious = $suspicious
        }
    } | Sort-Object Suspicious -Descending | Format-Table -AutoSize
```

Windows Services analysis (PowerShell):

```
# Get services with non-standard paths or unusual configurations
Get-WmiObject Win32_Service |
    Select-Object Name, DisplayName, State, StartMode, PathName, StartName |
    Where-Object {
        $_.PathName -notmatch "^C:\\Windows" -and
        $_.PathName -notmatch "^C:\\Program Files" -and
        $_.PathName -notmatch "^C:\\Program Files \(x86\)" -and
        $_.StartMode -eq "Auto" -and
        $_.State -eq "Running"
    } |
    Sort-Object Name |
    Format-Table -AutoSize
```

WMI persistence (PowerShell):

```
# Check for WMI event subscription persistence
Get-WmiObject -Namespace root\Subscription -Class __EventFilter
Get-WmiObject -Namespace root\Subscription -Class __EventConsumer
Get-WmiObject -Namespace root\Subscription -Class __FilterToConsumerBinding
```

Startup folder inspection (PowerShell):

```
# Check common startup folders
$startupFolders = @(
    "$env:APPDATA\Microsoft\Windows\Start Menu\Programs\Startup",
    "$env:ProgramData\Microsoft\Windows\Start
Menu\Programs\Startup"
)

foreach ($folder in $startupFolders) {
    if (Test-Path $folder) {
```

```
        Write-Host "Checking startup folder: $folder"
-ForegroundColor Cyan
        Get-ChildItem -Path $folder -Recurse |
            Select-Object FullName, CreationTime, LastWriteTime,
Length
    }
}
```

DLL Search Order Hijacking detection (PowerShell):

```
# Check for potential DLL hijacking in PATH directories
$paths = $env:PATH -split ";"
foreach ($path in $paths) {
    if (Test-Path $path) {
        $dlls = Get-ChildItem -Path $path -Filter "*.dll" -ErrorAction
SilentlyContinue
        if ($dlls) {
            Write-Host "Found DLLs in PATH directory: $path" -ForegroundColor
Yellow
            $dlls | Select-Object FullName, CreationTime, LastWriteTime |
                Format-Table -AutoSize
        }
    }
}
```

For a comprehensive persistence hunting approach, these techniques should be combined and automated in a script that runs regularly and reports any unusual findings or changes from a known-good baseline.

Conclusion

Windows and PowerShell commands provide cybersecurity professionals with powerful capabilities for securing, investigating, and responding to threats in Windows environments. From basic file system operations to advanced threat hunting techniques, mastering these command-line tools enables security practitioners to gain deep visibility into system operations and develop effective security controls.

The most effective security professionals leverage a combination of native Windows commands and PowerShell's more advanced scripting capabilities to automate security tasks, perform thorough investigations, and enhance defensive postures. By understanding both the capabilities and potential security implications of these powerful tools, security teams can better protect their Windows environments while efficiently responding to security incidents.

As attackers continue to evolve their techniques, maintaining proficiency with Windows and PowerShell commands will remain an essential skill for cybersecurity professionals. By implementing the techniques detailed in this guide, security teams can ensure they have the necessary tools and knowledge to defend against modern threats targeting Windows environments.

Resources and Further Reading

1. Microsoft PowerShell Security Documentation: https://docs.microsoft.com/en-us/powershell/scripting/security/overview
2. Windows Command Reference for Security Professionals: https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/windows-commands
3. PowerShell for Security Professionals (SANS Course): https://www.sans.org/cyber-security-courses/securing-windows-powershell/
4. MITRE ATT&CK Framework - Windows Persistence Techniques: https://attack.mitre.org/tactics/TA0003/
5. Windows Event Log Analysis Techniques: https://www.ultimatewindowssecurity.com/securitylog/encyclopedia/