```
Get-NetTCPConnection | Where-Object { $_.State -eq 'Listen' -and $_.LocalAddress -eq
'0.0.0.0' } | Select-Object LocalAddress, LocalPort, State
```


```
Get-NetTCPConnection | Where-Object { $_.State -eq 'Listen' -and $_.LocalAddress -eq
'0.0.0.0' } | Select-Object LocalAddress, LocalPort, State
```


```
    $shares = Get-NetworkShares

    if ($shares -eq $null) {
        Write-Host "No shares found or unable to retrieve shares."
        return
    }

    $unauthorizedShares = @()

    foreach ($share in $shares) {
        # Check if share name matches the unauthorized pattern
        if ($share.Name -like $UnauthorizedSharePattern) {
            $unauthorizedShares += $share
        }

        # Check if the share is in a sensitive folder (optional)
        if ($share.Path -like "$SensitivePath*") {
            $unauthorizedShares += $share
        }
```

```
    }

    # If unauthorized shares are found, display them
    if ($unauthorizedShares.Count -gt 0) {
        Write-Host "Unauthorized network shares found:"
        $unauthorizedShares | Format-Table Name, Path, Description
    } else {
        Write-Host "No unauthorized network shares detected."
    }
}

# Example usage
$unauthorizedSharePattern = "*public*"  # Example pattern for share names (e.g., any share with 'public' in the name)
$sensitivePath = "C:\SensitiveData"    # Example sensitive path

Check-UnauthorizedShares -UnauthorizedSharePattern $unauthorizedSharePattern -SensitivePath $sensitivePath
```

```
# Enable Windows Defender Firewall for all network profiles (Domain, Private, Public)
Write-Host "Enabling Windows Defender Firewall for all network profiles..."
Set-NetFirewallProfile -Profile Domain,Private,Public -Enabled True
Write-Host "Windows Defender Firewall is now enabled for all network profiles."
```

```
# Enable Windows Defender Firewall for all network profiles (Domain, Private, Public)
Write-Host "Enabling Windows Defender Firewall for all network profiles..."
Set-NetFirewallProfile -Profile Domain,Private,Public -Enabled True
Write-Host "Windows Defender Firewall is now
```

```
# Enable Windows Defender Firewall for all network profiles (Domain, Private, Public)
Write-Host "Enabling Windows Defender Firewall for all network profiles..."
Set-NetFirewallProfile -Profile Domain,Private,Public -Enabled True
Write-Host "Windows Defender Firewall is now enabled for all network profiles.”
```

```
# Enable Windows Defender Firewall for all network profiles (Domain, Private, Public)
Write-Host "Enabling Windows Defender Firewall for all network profiles..."
Set-NetFirewallProfile -Profile Domain,Private,Public -Enabled True
Write-Host "Windows Defender Firewall is now enabled for all network profiles.”
```

```
            Write-Host "WMI uninstall failed for $($program.Name). Trying alternative methods..."
        }
    }
}

# Try uninstall registry key method
$uninstallKeys = @(
    "HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall\*",
```

```
        "HKLM:\SOFTWARE\Wow6432Node\Microsoft\Windows\CurrentVersion\Uninstall\*"
    )

    $registryPrograms = Get-ItemProperty $uninstallKeys |
        Where-Object { $_.DisplayName -like "*$programName*" } |
        Select-Object UninstallString, DisplayName

    foreach ($regProgram in $registryPrograms) {
        if ($regProgram.UninstallString) {
            try {
                Write-Host "Attempting to remove $($regProgram.DisplayName) via registry uninstall
string..."
                Start-Process "cmd.exe" -ArgumentList "/c $($regProgram.UninstallString)" -Wait
-NoNewWindow
                Write-Host "$($regProgram.DisplayName) uninstalled via registry method."
            }
            catch {
                Write-Host "Failed to uninstall $($regProgram.DisplayName) via registry: $_"
            }
        }
    }
}

# Logging preparation
$logPath = "$env:TEMP\program_removal_log_$(Get-Date -Format 'yyyyMMdd_HHmmss').txt"
Start-Transcript -Path $logPath

# Loop through each program in the list and remove it
foreach ($program in $programsToRemove) {
    Remove-Program -programName $program
}

Stop-Transcript

Write-Host "Removal process completed. Check log at $logPath for details."


# Windows Server 2019 Backdoor Detection and Removal Script

Function Detect-Backdoors {
    # Network Listener Detection with Port Details
    $netstat = netstat -ano | Where-Object {
        $_ -match "LISTENING" -and
        $_ -notmatch "(127.0.0.1|::1)"
```

```
    } | ForEach-Object {
        $parts = $_.Trim() -split '\s+';
        [PSCustomObject]@{
            Protocol = $parts[0]
            LocalAddress = $parts[1]
            Port = ($parts[1] -split ':')[-1]
            State = $parts[3]
            ProcessID = $parts[4]
            ProcessName = (Get-Process -Id $parts[4] -ErrorAction SilentlyContinue).Name
        }
    }

    # Rest of the previous script remains the same...
    $results = [PSCustomObject]@{
        # Previous detection results...
        UnsafeNetworkListeners = $netstat
    }

    return $results
}

# Modify Remove-Backdoors function to log port details
Function Remove-Backdoors {
    param($backdoorDetectionResults)

    # Log Backdoor Port Details
    $backdoorDetectionResults.UnsafeNetworkListeners | ForEach-Object {
        Write-Host "Detected Backdoor: Port $($_.Port), Protocol $($_.Protocol), Process
$($_.ProcessName)" -ForegroundColor Red

        # Optional: Kill the process associated with the suspicious listener
        if ($_.ProcessID) {
            Stop-Process -Id $_.ProcessID -Force
        }
    }

    # Rest of the previous removal logic...
}

# Main Execution
$backdoors = Detect-Backdoors
Remove-Backdoors -backdoorDetectionResults $backdoors
```

```
# Disable FTP Service Script
# Requires Administrator Privileges

# Stop the FTP Service
Stop-Service -Name "FTPSVC" -Force

# Set the FTP Service Startup Type to Disabled
Set-Service -Name "FTPSVC" -StartupType Disabled

# Optional: Remove IIS FTP Feature (requires restart)
Uninstall-WindowsFeature -Name Web-Ftp-Server

# Verify Service Status
Get-Service -Name "FTPSVC"

Write-Host "FTP Service has been disabled and stopped."
```

```
# Script to Enable Blank Password Limitation Policy
# Requires Administrative Privileges

# Import Group Policy Module
Import-Module GroupPolicy

# Configure Local Group Policy
$policyPath = "Computer Configuration\Windows Settings\Security Settings\Local
Policies\Security Options"
$policyName = "Accounts: Limit local account use of blank passwords"

# Set the policy to Enabled
Set-GPRegistryValue -Name "Local Group Policy" -Key
"HKLM\SYSTEM\CurrentControlSet\Control\Lsa" -ValueName "LimitBlankPasswordUse" -Type
DWord -Value 1
```

```
# Force Group Policy Update
gpupdate /force

Write-Host "Blank password limitation policy has been enabled."
```

```
# Account Security Policy Configuration Script
# Requires Administrator Privileges

# Import Group Policy Module
Import-Module GroupPolicy

# Password Policy Configuration
secedit /configure /db secedit.sdb /cfg secedit.inf /Areas SECURITYPOLICY

# Set Specific Password Policy Parameters
net accounts /minpwlen:8
net accounts /maxpwage:90
net localgroup "Administrators" /maxpwage:30

# Advanced Password Policy Settings
$passwordPolicyPath = "HKLM:\SYSTEM\CurrentControlSet\Control\Password Policy"
Set-ItemProperty -Path $passwordPolicyPath -Name "PasswordHistorySize" -Value 5
Set-ItemProperty -Path $passwordPolicyPath -Name "MinPasswordAge" -Value 10
Set-ItemProperty -Path $passwordPolicyPath -Name "EnableComplexPasswords" -Value 1

# Disable Reversible Encryption
Set-ItemProperty -Path
"HKLM:\SYSTEM\CurrentControlSet\Services\LanmanWorkstation\Parameters" -Name
"EnablePlainTextPassword" -Value 0

# Account Lockout Policy Configuration
net accounts /lockoutduration:30
net accounts /lockoutthreshold:5
net accounts /lockoutwindow:30

# Force Group Policy Update
gpupdate /force

Write-Host "Account Security Policies have been configured successfully."
```

```
Function Detect-Backdoors {
    # Network Listener Detection with Port Details
    $netstat = netstat -ano | Where-Object {
        $_ -match "LISTENING" -and
        $_ -notmatch "(127.0.0.1|::1)"
    } | ForEach-Object {
        $parts = $_.Trim() -split '\s+'
        if ($parts.Length -ge 5) {
            $port = if ($parts[1] -match ":\d+") {
                ($parts[1] -split ':')[-1]
            } else {
                $null
            }
            [PSCustomObject]@{
                Protocol     = $parts[0]
                LocalAddress = $parts[1]
                Port         = $port
                State        = $parts[3]
                ProcessID    = $parts[4]
                ProcessName  = (Get-Process -Id $parts[4] -ErrorAction SilentlyContinue).Name
            }
        }
    }

    $results = [PSCustomObject]@{
        UnsafeNetworkListeners = $netstat
    }
```

```
    return $results
}

Function Remove-Backdoors {
    param($backdoorDetectionResults)

    $backdoorDetectionResults.UnsafeNetworkListeners | ForEach-Object {
        Write-Host "Detected Backdoor: Port $($_.Port), Protocol $($_.Protocol), Process
$($_.ProcessName)" -ForegroundColor Red

        $logMessage = "Detected Backdoor: Port $($_.Port), Protocol $($_.Protocol), Process
$($_.ProcessName)"
        $logMessage | Out-File "C:\backdoor_removal_log.txt" -Append

        if ($_.ProcessID) {
            Write-Host "Stopping Process $($_.ProcessName) with ID $($_.ProcessID)"
-ForegroundColor Yellow
            Stop-Process -Id $_.ProcessID -Force
        }
    }
}

# Main Execution Block
$backdoors = Detect-Backdoors
if ($backdoors.UnsafeNetworkListeners.Count -gt 0) {
    Remove-Backdoors -backdoorDetectionResults $backdoors
} else {
    Write-Host "No backdoors detected." -ForegroundColor Green
}




# Network Listener and Backdoor Detection Script

function Detect-SuspiciousListeners {
    # Get all TCP connections in Listen state on all interfaces
    $listeners = Get-NetTCPConnection |
        Where-Object {
            $_.State -eq 'Listen' -and
            $_.LocalAddress -eq '0.0.0.0'
        }
```

```
# Known safe ports list (common services)
$safePorts = @(
    ,    # SSH
    ,    # HTTP
    ,   # HTTPS
    ,  # RDP
    ,   # RPC
    ,   # SMB
    ,   # NetBIOS
        # DNS
)

# Detect potentially suspicious listeners
$suspiciousListeners = $listeners |
    Where-Object { $safePorts -notcontains $_.LocalPort }

# Analyze suspicious listeners
if ($suspiciousListeners) {
    Write-Host "POTENTIAL BACKDOOR DETECTED!" -ForegroundColor Red

    foreach ($listener in $suspiciousListeners) {
        # Get process information for the listener
        $process = Get-Process -Id $listener.OwningProcess

        Write-Host "Suspicious Listener Detected:" -ForegroundColor Yellow
        Write-Host "Port: $($listener.LocalPort)"
        Write-Host "Process: $($process.ProcessName)"
        Write-Host "Process Path: $($process.Path)"

        # Additional investigation steps
        if ($process) {
            # Check process integrity and origin
            $fileInfo = Get-Item $process.Path
            $signature = Get-AuthenticodeSignature $process.Path

            Write-Host "File Created: $($fileInfo.CreationTime)"
            Write-Host "Digital Signature Status: $($signature.Status)"
        }
    }
}
else {
    Write-Host "No suspicious listeners detected." -ForegroundColor Green
}
```

```
}

# Run the detection function
Detect-SuspiciousListeners
```

```
function Detect-UnauthorizedNetworkShares {
    param(
        [string]$UnauthorizedSharePattern = "*public*",
        [string]$SensitivePath = "C:\SensitiveData"
    )

    try {
        $shares = Get-SmbShare | Where-Object { $_.Name -ne '$ADMIN' }
    }
    catch {
        return
    }

    $unauthorizedShares = $shares | Where-Object {
        ($_.Name -like $UnauthorizedSharePattern) -or
        ($_.Path -like "$SensitivePath*")
    }

    if ($unauthorizedShares) {
        Write-Host "Unauthorized Network Shares Detected:" -ForegroundColor Red
        $unauthorizedShares | Format-Table Name, Path, Description
    }
    else {
        Write-Host "No unauthorized network shares found." -ForegroundColor Green
    }
}

Detect-UnauthorizedNetworkShares
```

```
# User Rights Configuration Script

function Configure-UserRights {
    # Access computer from network
    secedit /configure /db secedit.sdb /cfg secedit.inf /areas USER_RIGHTS
    $netAccessGroup = "Administrators", "Authenticated Users"

    # Disable 'Act as part of the operating system'
    $osActGroup = @()

    # Deny local logon and network access for Guest
    $denyLocalLogon = "Guest"
    $denyNetworkAccess = "Guest"

    # Take ownership restricted to Administrators
    $takeOwnershipGroup = "Administrators"

    # Apply configurations using local group policy
    net localgroup "Administrators" /add
    # Additional commands to apply specific user rights would be added here
}

Configure-UserRights
```

```
# Windows Update Configuration and Management Script
# Run as Administrator

# Function to Check and Configure Windows Update Service
function Configure-WindowsUpdate {
    try {
        # Stop Windows Update Service
        Stop-Service -Name "wuauserv" -Force -ErrorAction Stop

        # Set Windows Update Service to Automatic Startup
        Set-Service -Name "wuauserv" -StartupType Automatic -ErrorAction Stop

        # Start Windows Update Service
        Start-Service -Name "wuauserv" -ErrorAction Stop

        # Configure Update Settings
        $updateSettings = @{
            "NoAutoUpdate" = 0  # Enable Automatic Updates
            "AUOptions" = 4     # Auto download and schedule installation
        }

        # Set Registry Keys for Windows Update
        Set-ItemProperty -Path
"HKLM:\SOFTWARE\Policies\Microsoft\Windows\WindowsUpdate\AU" -Name "NoAutoUpdate"
-Value 0
        Set-ItemProperty -Path
"HKLM:\SOFTWARE\Policies\Microsoft\Windows\WindowsUpdate\AU" -Name "AUOptions"
-Value 4

        # Configure Active Hours (prevents updates during work hours)
        Set-ItemProperty -Path "HKLM:\SOFTWARE\Microsoft\WindowsUpdate\UX\Settings"
-Name "ActiveHoursStart" -Value 8
        Set-ItemProperty -Path "HKLM:\SOFTWARE\Microsoft\WindowsUpdate\UX\Settings"
-Name "ActiveHoursEnd" -Value 17

        # Force Windows Update Detection
        wuauclt.exe /detectnow

        Write-Host "Windows Update configured successfully!" -ForegroundColor Green
    }
    catch {
        Write-Host "Error configuring Windows Update: $_" -ForegroundColor Red
    }
}
```

```
# Function to Check Update Status
function Check-WindowsUpdateStatus {
    try {
        $updateService = Get-Service -Name "wuauserv"
        $updateSession = New-Object -ComObject Microsoft.Update.Session
        $updateSearcher = $updateSession.CreateUpdateSearcher()

        Write-Host "Windows Update Service Status:" -ForegroundColor Cyan
        Write-Host "------------------------------"
        Write-Host "Service Name: $($updateService.Name)"
        Write-Host "Service Status: $($updateService.Status)"
        Write-Host "Startup Type: $($updateService.StartType)"

        # Pending Updates Check
        $pendingUpdates = $updateSearcher.Search("IsInstalled=0")
        Write-Host "Pending Updates: $($pendingUpdates.Updates.Count)" -ForegroundColor
Yellow
    }
    catch {
        Write-Host "Unable to retrieve Windows Update details: $_" -ForegroundColor Red
    }
}

# Main Execution
Write-Host "Configuring Windows Update..." -ForegroundColor Cyan
Configure-WindowsUpdate
Start-Sleep -Seconds 5
Check-WindowsUpdateStatus

# Optional: Trigger Windows Update Download
function Trigger-WindowsUpdateDownload {
    try {
        # Initiate Windows Update Download
        $updateSession = New-Object -ComObject Microsoft.Update.Session
        $updateSearcher = $updateSession.CreateUpdateSearcher()
        $searchResult = $updateSearcher.Search("IsInstalled=0 and Type='Software'")

        if ($searchResult.Updates.Count -gt 0) {
            $updatesToDownload = New-Object -ComObject Microsoft.Update.UpdateColl
            foreach ($update in $searchResult.Updates) {
                $updatesToDownload.Add($update) | Out-Null
            }
```

```
            $downloader = $updateSession.CreateUpdateDownloader()
            $downloader.Updates = $updatesToDownload
            $downloader.Download()

            Write-Host "Updates downloaded successfully!" -ForegroundColor Green
        }
        else {
            Write-Host "No updates available for download." -ForegroundColor Yellow
        }
    }
    catch {
        Write-Host "Error downloading updates: $_" -ForegroundColor Red
    }
}

# Uncomment the line below if you want to automatically download updates
# Trigger-WindowsUpdateDownload

Write-Host "Windows Update Configuration Complete!" -ForegroundColor Green
```

```
# Lock Screen Timeout Enforcement Script
# Run as Administrator

# Global Variables
$timeoutSeconds = 600  # 10 minutes
$registryPath = "HKCU:\Control Panel\Desktop"
$gpolicyPath = "HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\System"

# Function to Set Screen Timeout Settings
function Set-LockScreenTimeout {
    try {
        # User-Level Screen Saver Settings
        Set-ItemProperty -Path $registryPath -Name "ScreenSaveActive" -Value 1
```

```
        Set-ItemProperty -Path $registryPath -Name "ScreenSaveTimeOut" -Value
$timeoutSeconds

        # Group Policy Level Lock Screen Settings
        Set-ItemProperty -Path $gpolicyPath -Name "InactivityTimeoutSecs" -Value
$timeoutSeconds
        Set-ItemProperty -Path $gpolicyPath -Name "DisableLockScreenSlideShow" -Value 1

        # Windows 10/11 Lock Screen Policies
        New-ItemProperty -Path $gpolicyPath -Name "EnableLockScreenAppNotifications" -Value
0 -PropertyType DWord -Force
    }
    catch {
        Write-Host "Error setting lock screen timeout: $_" -ForegroundColor Red
    }
}

# Function to Disable Sleep and Hibernation
function Disable-SleepSettings {
    try {
        # Disable Sleep on AC Power
        powercfg /change standby-timeout-ac 0

        # Disable Sleep on Battery
        powercfg /change standby-timeout-dc 0

        # Disable Hibernation
        powercfg /hibernate off
    }
    catch {
        Write-Host "Error configuring power settings: $_" -ForegroundColor Red
    }
}

# Function to Configure Screen Timeout Policies
function Configure-ScreenTimeoutPolicies {
    try {
        # Advanced Group Policy Settings for Screen Lock
        $gpUpdate = @{

"Machine\System\CurrentControlSet\Control\Power\PowerSettings\ActivePowerScheme" =
"Enforce"
            "User\Software\Policies\Microsoft\Windows\Control
Panel\Desktop\ScreenSaveTimeOut" = $timeoutSeconds
```

```
        }

        # Modify Local Group Policy
        Local-GPO-Modify
    }
    catch {
        Write-Host "Error configuring screen timeout policies: $_" -ForegroundColor Red
    }
}

# Function to Verify Lock Screen Settings
function Test-LockScreenConfiguration {
    $currentTimeout = (Get-ItemProperty -Path $registryPath -Name
"ScreenSaveTimeOut").ScreenSaveTimeOut
    $screenSaverActive = (Get-ItemProperty -Path $registryPath -Name
"ScreenSaveActive").ScreenSaveActive

    Write-Host "Lock Screen Configuration Report:" -ForegroundColor Cyan
    Write-Host "---------------------------------"
    Write-Host "Current Timeout: $currentTimeout seconds" -ForegroundColor Green
    Write-Host "Screen Saver Active: $(if($screenSaverActive -eq 1){'Yes'}else{'No'})"
-ForegroundColor Yellow
}

# Main Execution
function Invoke-LockScreenEnforcement {
    Write-Host "Enforcing Lock Screen Security Settings..." -ForegroundColor Cyan

    # Execute Configuration Functions
    Set-LockScreenTimeout
    Disable-SleepSettings
    Configure-ScreenTimeoutPolicies

    # Verify Settings
    Test-LockScreenConfiguration

    Write-Host "Lock Screen Security Enforcement Complete!" -ForegroundColor Green
}

# Run the enforcement
Invoke-LockScreenEnforcement
```

```
# Backdoor Detection and Removal Script
# For Windows Server 2019 and 2022
# Run with Administrator Privileges

function Invoke-BackdoorScanner {
  param (
    [switch]$Detailed = $false,
    [switch]$RemoveThreats = $false
  )

  $threatDetected = $false
  $threats = @()

  Write-Host "Starting Comprehensive Backdoor Detection..." -ForegroundColor Cyan

  function Detect-SuspiciousServices {
    $suspiciousServices = Get-Service | Where-Object {
      $_.Name -match "^(win|sys|remote)" -or
      $_.DisplayName -match "unknown|hidden|backdoor" -or
      $_.Status -eq 'Running' -and $_.StartType -eq 'Disabled'
    }

    if ($suspiciousServices) {
      $threatDetected = $true
      $threats += @{
        Type = "Suspicious Services"
        Details = $suspiciousServices | Select-Object Name, DisplayName, Status
      }

      if ($RemoveThreats) {
        $suspiciousServices | Stop-Service -Force
        $suspiciousServices | Set-Service -StartupType Disabled
      }
    }
  }
```

```
function Detect-SuspiciousScheduledTasks {
    $suspiciousTasks = Get-ScheduledTask | Where-Object {
        $_.TaskName -match "^(hidden|backdoor|payload)" -or
        $_.Principal.UserId -eq 'S-1-5-21-0-0-0-0000'
    }

    if ($suspiciousTasks) {
        $threatDetected = $true
        $threats += @{
            Type = "Suspicious Scheduled Tasks"
            Details = $suspiciousTasks | Select-Object TaskName, State
        }

        if ($RemoveThreats) {
            $suspiciousTasks | Unregister-ScheduledTask -Confirm:$false
        }
    }
}

function Detect-SuspiciousRegistryEntries {
    $suspiciousRegKeys = @(
        "HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\Run",
        "HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnce",
        "HKCU:\SOFTWARE\Microsoft\Windows\CurrentVersion\Run"
    )

    $suspiciousEntries = $suspiciousRegKeys | ForEach-Object {
        Get-ItemProperty $_ | Where-Object {
            $_.PSPath -notmatch "^(Microsoft|Windows)"
        }
    }

    if ($suspiciousEntries) {
        $threatDetected = $true
        $threats += @{
            Type = "Suspicious Registry Autorun Entries"
            Details = $suspiciousEntries
        }

        if ($RemoveThreats) {
            $suspiciousEntries | Remove-ItemProperty -Force
        }
    }
```

```
}

function Detect-UnusualNetworkListeners {
    $unusualListeners = Get-NetTCPConnection | Where-Object {
        $_.State -eq 'Listen' -and
        $_.LocalPort -notin (80,443,3389,22,21) -and
        $_.RemoteAddress -ne '::' -and
        $_.RemoteAddress -ne '0.0.0.0'
    }

    if ($unusualListeners) {
        $threatDetected = $true
        $threats += @{
            Type = "Unusual Network Listeners"
            Details = $unusualListeners | Select-Object LocalPort, RemoteAddress
        }
    }
}

function Detect-SuspiciousPowerShellProfiles {
    $profilePaths = @(
        "$env:USERPROFILE\Documents\WindowsPowerShell\profile.ps1",
        "$env:USERPROFILE\Documents\PowerShell\profile.ps1"
    )

    $suspiciousProfiles = $profilePaths | Where-Object {
        Test-Path $_ -and
        (Get-Content $_ | Select-String -Pattern "Invoke-|IEX|FromBase64String")
    }

    if ($suspiciousProfiles) {
        $threatDetected = $true
        $threats += @{
            Type = "Suspicious PowerShell Profiles"
            Details = $suspiciousProfiles
        }

        if ($RemoveThreats) {
            $suspiciousProfiles | Remove-Item -Force
        }
    }
}

Detect-SuspiciousServices
```

```
    Detect-SuspiciousScheduledTasks
    Detect-SuspiciousRegistryEntries
    Detect-UnusualNetworkListeners
    Detect-SuspiciousPowerShellProfiles

    if ($threatDetected) {
        Write-Host "[!] POTENTIAL BACKDOORS DETECTED!" -ForegroundColor Red
        if ($Detailed) {
            $threats | Format-Table -AutoSize
        }
    }
    else {
        Write-Host "[✓] No Backdoors Detected" -ForegroundColor Green
    }

    if ($Detailed) {
        $threats | Export-Csv -Path "$env:TEMP\backdoor_scan_report.csv" -NoTypeInformation
        Write-Host "Detailed report saved to $env:TEMP\backdoor_scan_report.csv"
-ForegroundColor Yellow
    }
}

Invoke-BackdoorScanner -Detailed
```

```
function Invoke-TrojanScanner {
  param (
    [switch]$Detailed = $false,
    [switch]$RemoveTrojans = $false
  )

  $trojanDetected = $false
  $trojans = @()

  Write-Host "Initiating Comprehensive Trojan Detection..." -ForegroundColor Cyan

  $suspiciousExtensions = @('.exe', '.dll', '.bat', '.cmd', '.vbs', '.ps1')

  $highRiskDirectories = @(
    "$env:TEMP",
    "$env:USERPROFILE\Downloads",
    "C:\Windows\Temp"
  )

  function Scan-SuspiciousFiles {
    foreach ($dir in $highRiskDirectories) {
      $suspiciousFiles = Get-ChildItem -Path $dir -Recurse -Include $suspiciousExtensions |
Where-Object {
        $_.Length -gt 0 -and
        $_.Name -match "(hack|inject|trojan|payload|exploit)" -or
        $_.CreationTime -lt (Get-Date).AddDays(-30)
      }

      if ($suspiciousFiles) {
        $trojanDetected = $true
        $trojans += @{
          Type = "Suspicious Files"
          Location = $dir
          Files = $suspiciousFiles | Select-Object FullName, Length, CreationTime
        }
```

```
            if ($RemoveTrojans) {
                $suspiciousFiles | Remove-Item -Force
            }
        }
    }
}

function Check-UnusualProcesses {
    $suspiciousProcesses = Get-Process | Where-Object {
        $_.Name -match "^(unknown|hidden|remote)" -or
        $_.Path -match "($env:TEMP|Downloads)" -or
        $_.CPU -gt 90
    }

    if ($suspiciousProcesses) {
        $trojanDetected = $true
        $trojans += @{
            Type = "Suspicious Processes"
            Details = $suspiciousProcesses | Select-Object ProcessName, Id, CPU
        }

        if ($RemoveTrojans) {
            $suspiciousProcesses | Stop-Process -Force
        }
    }
}

function Scan-RegistryPersistence {
    $persistenceKeys = @(
        "HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\Run",
        "HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnce",
        "HKCU:\SOFTWARE\Microsoft\Windows\CurrentVersion\Run"
    )

    $suspiciousEntries = $persistenceKeys | ForEach-Object {
        Get-ItemProperty $_ | Where-Object {
            $_.PSPath -notmatch "^(Microsoft|Windows)" -and
            $_.PSPath -match "($env:TEMP|Downloads)"
        }
    }

    if ($suspiciousEntries) {
        $trojanDetected = $true
```

```
            $trojans += @{
                Type = "Registry Persistence"
                Details = $suspiciousEntries
            }

            if ($RemoveTrojans) {
                $suspiciousEntries | Remove-ItemProperty -Force
            }
        }
    }

    function Analyze-NetworkConnections {
        $suspiciousConnections = Get-NetTCPConnection | Where-Object {
            $_.RemoteAddress -notmatch "^(127\.|::1|0\.0\.0\.0)" -and
            $_.RemotePort -notin (80,443,22,3389)
        }

        if ($suspiciousConnections) {
            $trojanDetected = $true
            $trojans += @{
                Type = "Unusual Network Connections"
                Details = $suspiciousConnections | Select-Object LocalPort, RemoteAddress,
RemotePort
            }
        }
    }

    Scan-SuspiciousFiles
    Check-UnusualProcesses
    Scan-RegistryPersistence
    Analyze-NetworkConnections

    if ($trojanDetected) {
        Write-Host "[!] POTENTIAL TROJANS DETECTED!" -ForegroundColor Red
        if ($Detailed) {
            $trojans | Format-Table -AutoSize
        }
    }
    else {
        Write-Host "[✓] No Trojans Detected" -ForegroundColor Green
    }

    if ($Detailed) {
        $trojans | Export-Csv -Path "$env:TEMP\trojan_scan_report.csv" -NoTypeInformation
```

```
    Write-Host "Detailed report saved to $env:TEMP\trojan_scan_report.csv" -ForegroundColor
Yellow
  }
}

Invoke-TrojanScanner -Detailed
```