WINDOWS EVENT LOG ANALYSIS

ADVANCED THREAT DETECTION & INVESTIGATION IN ENTERPRISE SECURITY

OKAN YILDIZ

SEPTEMBER 2025

Executive Summary

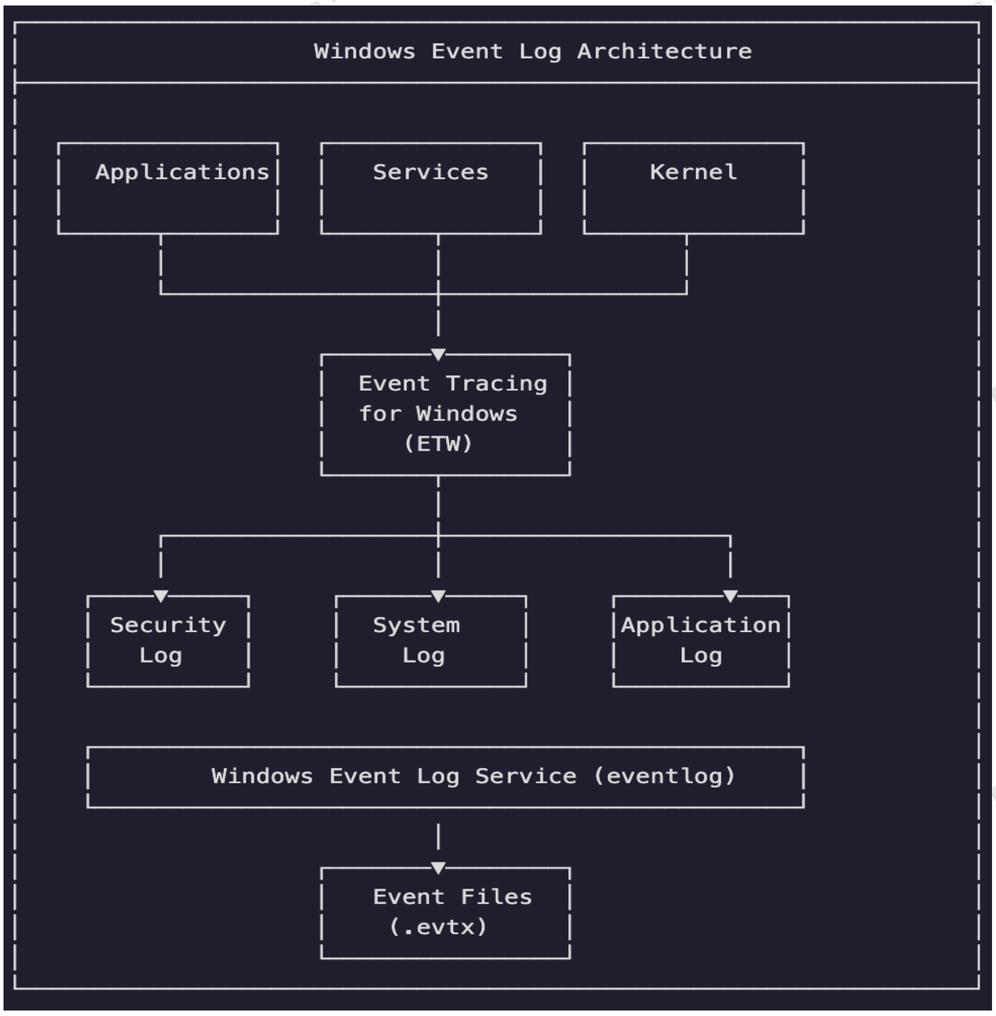
Windows Event Logs serve as the digital forensic backbone of enterprise security operations, capturing every system activity, authentication attempt, and security-relevant action across Windows infrastructure. **Mastering event log analysis transforms raw data into actionable threat intelligence**, enabling security teams to detect sophisticated attacks, investigate incidents, and maintain regulatory compliance.

This comprehensive guide explores advanced Windows Event Log analysis techniques, from understanding the underlying architecture to implementing automated threat detection systems. We'll cover critical security event IDs, correlation strategies, and practical investigation workflows that security professionals need to identify and respond to modern cyber threats effectively.

Understanding Windows Event Log Architecture

Event Log Fundamentals

Windows Event Logs operate through a sophisticated architecture that captures system, application, and security activities across the enterprise:


Windows Event Log Architecture
Applications
Services
Kernel
Event Tracing for Windows (ETW)
Security Log
System Log
Application Log
Windows Event Log Service (eventlog)
Event Files (.evtx)

Core Event Log Categories

Windows maintains several critical log categories:

Log Category	Primary Purpose	Default Location	Key Security Events
Security	Authentication, authorization, audit events	`%SystemRoot%\System32\Winevt\Logs\Security.evtx`	4624, 4625, 4648, 4672
System	System component events, driver issues	`%SystemRoot%\System32\Winevt\Logs\System.evtx`	7045, 7040, 1074
Application	Application-specific events	`%SystemRoot%\System32\Winevt\Logs\Application.evtx`	1000, 1001, 1002
PowerShell/Operational	PowerShell command execution	`Microsoft-Windows-PowerShell%4Operational.evtx`	4103, 4104, 4105
Sysmon	Detailed system activity monitoring	`Microsoft-Windows-Sysmon%4Operational.evtx`	1, 3, 7, 10, 11
Windows Defender	Antimalware activities	`Microsoft-Windows-Windows Defender%4Operational.evtx`	1116, 1117, 5001

Critical Security Event IDs for Threat Detection

Authentication and Logon Events

Understanding authentication patterns is crucial for detecting unauthorized access:

```
powershell
# PowerShell script to analyze authentication events
```

```powershell
function Analyze-AuthenticationEvents {
    param(
        [string]$ComputerName = $env:COMPUTERNAME,
        [int]$Hours = 24
    )

    $StartTime = (Get-Date).AddHours(-$Hours)

    # Critical authentication event IDs
    $AuthEventIDs = @{
        4624 = "Successful Logon"
        4625 = "Failed Logon"
        4634 = "Logoff"
        4648 = "Explicit Credential Logon"
        4672 = "Special Privileges Assigned"
        4768 = "Kerberos TGT Request"
        4769 = "Kerberos Service Ticket Request"
        4771 = "Kerberos Pre-authentication Failed"
        4776 = "NTLM Authentication"
    }

    $Results = @()

    foreach ($EventID in $AuthEventIDs.Keys) {
        $Events = Get-WinEvent -FilterHashtable @{
            LogName = 'Security'
            ID = $EventID
            StartTime = $StartTime
        } -ComputerName $ComputerName -ErrorAction SilentlyContinue

        if ($Events) {
            $EventAnalysis = $Events | Group-Object -Property {
                $_.Properties[5].Value  # Account Name
            } | Select-Object @{
                Name = 'EventType'
                Expression = {$AuthEventIDs[$EventID]}
            }, @{
                Name = 'Account'
                Expression = {$_.Name}
            }, Count | Sort-Object Count -Descending

            $Results += $EventAnalysis
        }
    }

    return $Results | Format-Table -AutoSize
}

# Example usage
```

Analyze-AuthenticationEvents -Hours 48

Privilege Escalation Indicators

Key events indicating potential privilege escalation:

Event ID	Description	Security Significance	Investigation Priority
4672	Special privileges assigned to new logon	Administrative access granted	**Critical**
4673	Privileged service called	Sensitive privilege use	**High**
4674	Operation attempted on privileged object	Potential privilege abuse	**High**
4688	New process created (with token elevation)	Process creation with elevated privileges	**Medium**
4703	Token right adjusted	User rights modification	**High**
4728	Member added to security-enabled global group	Group membership changes	**Critical**
4732	Member added to security-enabled local group	Local admin additions	**Critical**
4756	Member added to security-enabled universal group	Domain-wide privilege changes	**Critical**

Advanced Threat Detection Patterns

Lateral Movement Detection

Identifying lateral movement requires correlating multiple event sources:

```
powershell
# Advanced lateral movement detection script
function Detect-LateralMovement {
    [CmdletBinding()]
```

```powershell
param(
    [DateTime]$StartTime = (Get-Date).AddHours(-24),
    [string[]]$Computers = @($env:COMPUTERNAME)
)

$LateralMovementIndicators = @{
    # Network logons from unusual sources
    NetworkLogons = @{
        EventID = 4624
        LogonType = 3  # Network logon
    }

    # Explicit credential usage
    ExplicitCredentials = @{
        EventID = 4648
    }

    # Remote Desktop connections
    RDPConnections = @{
        EventID = @(4624, 4778, 4779)
        LogonType = 10  # RemoteInteractive
    }

    # Service installations (PsExec-like behavior)
    ServiceInstallations = @{
        EventID = 7045
        LogName = 'System'
    }

    # WMI Activity
    WMIActivity = @{
        EventID = 5857
        LogName = 'Microsoft-Windows-WMI-Activity/Operational'
    }

    # PowerShell Remoting
    PSRemoting = @{
        EventID = @(4103, 4104)
        LogName = 'Microsoft-Windows-PowerShell/Operational'
    }
}

$DetectedMovements = @()

foreach ($Computer in $Computers) {
    Write-Host "Analyzing $Computer for lateral movement..."
```

```
-ForegroundColor Cyan

        # Check for network logons with suspicious patterns
        $NetworkLogons = Get-WinEvent -FilterHashtable @{
            LogName = 'Security'
            ID = 4624
            StartTime = $StartTime
        } -ComputerName $Computer -ErrorAction SilentlyContinue |
        Where-Object {
            $_.Properties[8].Value -eq 3 -and  # Network logon
            $_.Properties[5].Value -notlike "*$" -and  # Not computer
account
            $_.Properties[18].Value -ne '-' -and  # Has source IP
            $_.Properties[18].Value -notmatch '^(127\.|::1)'  # Not
localhost
        }

        if ($NetworkLogons) {
            $DetectedMovements += [PSCustomObject]@{
                Computer = $Computer
                Type = "Network Logon"
                Count = $NetworkLogons.Count
                UniqueAccounts = ($NetworkLogons | ForEach-Object
{$_.Properties[5].Value} |
                                 Select-Object -Unique).Count
                SourceIPs = $NetworkLogons | ForEach-Object
{$_.Properties[18].Value} |
                           Select-Object -Unique
            }
        }

        # Check for service installations
        $ServiceInstalls = Get-WinEvent -FilterHashtable @{
            LogName = 'System'
            ID = 7045
            StartTime = $StartTime
        } -ComputerName $Computer -ErrorAction SilentlyContinue

        if ($ServiceInstalls) {
            foreach ($Event in $ServiceInstalls) {
                $ServiceName = $Event.Properties[0].Value
                $ServicePath = $Event.Properties[1].Value

                # Check for suspicious service patterns
                if ($ServicePath -match
'cmd\.exe|powershell\.exe|psexec|wmi') {
```

```
                    $DetectedMovements += [PSCustomObject]@{
                        Computer = $Computer
                        Type = "Suspicious Service Installation"
                        ServiceName = $ServiceName
                        ServicePath = $ServicePath
                        TimeCreated = $Event.TimeCreated
                    }
                }
            }
        }
    }

    return $DetectedMovements
}
```

Persistence Mechanism Detection

Comprehensive persistence detection across Windows systems:

```
powershell
# Persistence mechanism detection framework
function Find-PersistenceMechanisms {
    param(
        [string]$ComputerName = $env:COMPUTERNAME,
        [int]$DaysBack = 7
    )

    $PersistenceEvents = @{
        # Registry Run Keys
        RegistryPersistence = @{
            EventIDs = @(4657, 4663)
            Patterns = @(
                'Run',
                'RunOnce',
                'RunServices',
                'RunServicesOnce',
                'Userinit',
                'Shell',
                'AppInit_DLLs'
            )
        }

        # Scheduled Tasks
        ScheduledTasks = @{
            EventIDs = @(4698, 4699, 4700, 4701, 4702)
            LogName = 'Security'
        }

        # Service Creation/Modification
        Services = @{
            EventIDs = @(4697, 7045, 7040)
            LogName = @('Security', 'System')
        }

        # WMI Event Subscriptions
        WMIPersistence = @{
            EventIDs = @(5857, 5858, 5859, 5860, 5861)
            LogName = 'Microsoft-Windows-WMI-Activity/Operational'
        }

        # Startup Folder Modifications
        StartupFolder = @{
            EventIDs = @(4663, 4656)
            Patterns = @(
                'Startup',
                'Start Menu'
            )
        }
    }

    $Results = @()
    $StartTime = (Get-Date).AddDays(-$DaysBack)

    # Check for registry-based persistence
    Write-Host "Checking Registry Persistence..." -ForegroundColor
Yellow
    $RegEvents = Get-WinEvent -FilterHashtable @{
        LogName = 'Security'
        ID = 4657
        StartTime = $StartTime
    } -ComputerName $ComputerName -ErrorAction SilentlyContinue |
    Where-Object {
        $ObjectName = $_.Properties[5].Value
        $PersistenceEvents.RegistryPersistence.Patterns | ForEach-Object
{
            if ($ObjectName -match $_) { return $true }
        }
        return $false
    }

    if ($RegEvents) {
        $Results += [PSCustomObject]@{
            Type = "Registry Persistence"
            Count = $RegEvents.Count
            Details = $RegEvents | Select-Object TimeCreated,
                @{N='ObjectName';E={$_.Properties[5].Value}},
                @{N='ProcessName';E={$_.Properties[11].Value}}
        }
    }

    # Check for scheduled task creation
    Write-Host "Checking Scheduled Tasks..." -ForegroundColor Yellow
    $TaskEvents = Get-WinEvent -FilterHashtable @{
        LogName = 'Microsoft-Windows-TaskScheduler/Operational'
        ID = @(106, 140, 141)  # Task registered, updated, deleted
        StartTime = $StartTime
    } -ComputerName $ComputerName -ErrorAction SilentlyContinue

    if ($TaskEvents) {
        $Results += [PSCustomObject]@{
            Type = "Scheduled Task Activity"
            Count = $TaskEvents.Count
            RecentTasks = $TaskEvents | Select-Object -First 10
TimeCreated, Message
        }
    }

    # Check for new services
    Write-Host "Checking Service Installations..." -ForegroundColor
Yellow
    $ServiceEvents = Get-WinEvent -FilterHashtable @{
        LogName = 'System'
        ID = 7045
        StartTime = $StartTime
    } -ComputerName $ComputerName -ErrorAction SilentlyContinue

    if ($ServiceEvents) {
        $SuspiciousServices = $ServiceEvents | Where-Object {
            $_.Message -match
'powershell|cmd|wscript|cscript|rundll32|regsvr32'
        }

        if ($SuspiciousServices) {
            $Results += [PSCustomObject]@{
                Type = "Suspicious Service Installation"
                Count = $SuspiciousServices.Count
                Services = $SuspiciousServices | Select-Object
TimeCreated,
                    @{N='ServiceName';E={$_.Properties[0].Value}},
                    @{N='ImagePath';E={$_.Properties[1].Value}}
            }
        }
    }

    return $Results
}
```

Event Log Correlation and Timeline Analysis

Building Attack Timelines

Creating comprehensive attack timelines requires correlating events across multiple logs:

```
powershell
# Advanced timeline correlation engine
function Build-SecurityTimeline {
    param(
        [DateTime]$StartTime,
        [DateTime]$EndTime,
        [string[]]$EventSources = @('Security', 'System',
'Application'),
        [string]$ExportPath = "SecurityTimeline.csv"
    )

    $Timeline = @()

    # Define critical security events for timeline
    $CriticalEvents = @{
        'Security' = @(
            1102,  # Audit log cleared
            4624,  # Successful logon
            4625,  # Failed logon
            4648,  # Explicit credentials
            4672,  # Special privileges
            4688,  # Process creation
            4697,  # Service installed
            4698,  # Scheduled task created
            4720,  # User account created
            4728,  # Member added to global group
            4732,  # Member added to local group
            4756,  # Member added to universal group
            4794,  # Directory service restore mode
            4964   # Special groups assigned
        )

        'System' = @(
            7045,  # Service installed
            7040,  # Service start type changed
            7036,  # Service started/stopped
            1074,  # System shutdown
            6005,  # Event log service started
            6006   # Event log service stopped
        )

        'Application' = @(
            1000,  # Application error
            1001,  # Application hang
            1002   # Application hang recovery
        )
    }

    # Collect events from each source
    foreach ($LogName in $EventSources) {
        if ($CriticalEvents.ContainsKey($LogName)) {
            Write-Host "Processing $LogName log..." -ForegroundColor
Green

            $Events = Get-WinEvent -FilterHashtable @{
                LogName = $LogName
                ID = $CriticalEvents[$LogName]
                StartTime = $StartTime
                EndTime = $EndTime
            } -ErrorAction SilentlyContinue

            foreach ($Event in $Events) {
                $Timeline += [PSCustomObject]@{
                    TimeGenerated = $Event.TimeCreated
                    LogSource = $LogName
                    EventID = $Event.Id
                    Level = $Event.LevelDisplayName
                    Message = $Event.Message -replace '\r\n', ' '
-replace '\n', ' '
                    Computer = $Event.MachineName
                    UserAccount = if ($Event.UserId) {
                        try {

([System.Security.Principal.SecurityIdentifier]$Event.UserId).Translate(
[System.Security.Principal.NTAccount]).Value
```

```
                        } catch { $Event.UserId }
                    } else { "N/A" }
                }
            }
        }
    }

    # Add PowerShell events if available
    $PSLogs = @('Microsoft-Windows-PowerShell/Operational', 'Windows
PowerShell')
    foreach ($PSLog in $PSLogs) {
        try {
            $PSEvents = Get-WinEvent -FilterHashtable @{
                LogName = $PSLog
                StartTime = $StartTime
                EndTime = $EndTime
            } -ErrorAction SilentlyContinue |
            Where-Object { $_.Id -in @(4103, 4104, 4105, 4106, 400, 403)
}

            foreach ($Event in $PSEvents) {
                $Timeline += [PSCustomObject]@{
                    TimeGenerated = $Event.TimeCreated
                    LogSource = "PowerShell"
                    EventID = $Event.Id
                    Level = $Event.LevelDisplayName
                    Message = ($Event.Message -split "`n")[0]  # First
line only
                    Computer = $Event.MachineName
                    UserAccount = "PowerShell Execution"
                }
            }
        } catch {
            Write-Warning "Could not access PowerShell logs: $_"
        }
    }

    # Sort timeline chronologically
    $Timeline = $Timeline | Sort-Object TimeGenerated

    # Export to CSV
    $Timeline | Export-Csv -Path $ExportPath -NoTypeInformation

    # Display summary
    Write-Host "`nTimeline Summary:" -ForegroundColor Cyan
    Write-Host "Total Events: $($Timeline.Count)"
    Write-Host "Time Range: $($Timeline[0].TimeGenerated) to
$($Timeline[-1].TimeGenerated)"
    Write-Host "Exported to: $ExportPath"

    # Return timeline for further analysis
    return $Timeline
}
```

Cross-System Correlation

Enterprise-wide event correlation for detecting distributed attacks:

```
powershell
# Multi-system correlation framework
function Correlate-MultiSystemEvents {
    param(
        [string[]]$DomainControllers,
        [string[]]$MemberServers,
        [string]$SuspiciousAccount,
        [DateTime]$StartTime = (Get-Date).AddHours(-4)
    )

    $CorrelationResults = @{
        AuthenticationChain = @()
        PrivilegeEscalation = @()
        LateralMovement = @()
        DataAccess = @()
        Persistence = @()
    }

    # Track authentication flow across systems
    foreach ($DC in $DomainControllers) {
        Write-Host "Analyzing Domain Controller: $DC" -ForegroundColor
Yellow

        # Kerberos TGT requests
        $TGTEvents = Get-WinEvent -FilterHashtable @{
            LogName = 'Security'
            ID = 4768
            StartTime = $StartTime
        } -ComputerName $DC -ErrorAction SilentlyContinue |
        Where-Object { $_.Message -match $SuspiciousAccount }

        if ($TGTEvents) {
            $CorrelationResults.AuthenticationChain +=
[PSCustomObject]@{
                System = $DC
                EventType = "Kerberos TGT Request"
                Time = $TGTEvents[0].TimeCreated
                SourceIP = ($TGTEvents[0].Message |
                           Select-String -Pattern 'Client
Address:\s+::ffff:([^\s]+)' |
                           ForEach-Object {
$_.Matches[0].Groups[1].Value })
            }
        }

        # Service ticket requests
        $ServiceTickets = Get-WinEvent -FilterHashtable @{
            LogName = 'Security'
            ID = 4769
            StartTime = $StartTime
        } -ComputerName $DC -ErrorAction SilentlyContinue |
        Where-Object { $_.Message -match $SuspiciousAccount }

        foreach ($Ticket in $ServiceTickets) {
            $ServiceName = ($Ticket.Message |
                           Select-String -Pattern 'Service
Name:\s+([^\s]+)' |
                           ForEach-Object {
$_.Matches[0].Groups[1].Value })

            $CorrelationResults.LateralMovement += [PSCustomObject]@{
                System = $DC
                EventType = "Service Ticket Request"
                Time = $Ticket.TimeCreated
                TargetService = $ServiceName
                Account = $SuspiciousAccount
            }
        }
    }

    # Check member servers for actual access
    foreach ($Server in $MemberServers) {
        Write-Host "Analyzing Member Server: $Server" -ForegroundColor
Yellow

        # Network logons
        $NetworkLogons = Get-WinEvent -FilterHashtable @{
            LogName = 'Security'
            ID = 4624
            StartTime = $StartTime
        } -ComputerName $Server -ErrorAction SilentlyContinue |
        Where-Object {
            $_.Properties[8].Value -eq 3 -and  # Network logon
            $_.Message -match $SuspiciousAccount
        }

        if ($NetworkLogons) {
            $CorrelationResults.LateralMovement += [PSCustomObject]@{
                System = $Server
                EventType = "Network Logon"
                Time = $NetworkLogons[0].TimeCreated
                LogonType = "Type 3 (Network)"
                Account = $SuspiciousAccount
            }
        }

        # Check for privilege escalation
        $PrivEvents = Get-WinEvent -FilterHashtable @{
            LogName = 'Security'
            ID = 4672
            StartTime = $StartTime
        } -ComputerName $Server -ErrorAction SilentlyContinue |
        Where-Object { $_.Message -match $SuspiciousAccount }

        if ($PrivEvents) {
            $Privileges = ($PrivEvents[0].Message |
                          Select-String -Pattern 'Privileges:\s+(.+)'
-AllMatches |
                          ForEach-Object { $_.Matches[0].Groups[1].Value
})

            $CorrelationResults.PrivilegeEscalation +=
[PSCustomObject]@{
                System = $Server
                EventType = "Special Privileges Assigned"
                Time = $PrivEvents[0].TimeCreated
                Privileges = $Privileges
                Account = $SuspiciousAccount
            }
        }
    }

    # Generate correlation report
    Write-Host "`n=== CORRELATION ANALYSIS RESULTS ===" -ForegroundColor
Cyan
```

```
    # Show authentication flow
    if ($CorrelationResults.AuthenticationChain) {
        Write-Host "`nAuthentication Chain:" -ForegroundColor Green
        $CorrelationResults.AuthenticationChain |
            Sort-Object Time |
            Format-Table -AutoSize
    }

    # Show lateral movement pattern
    if ($CorrelationResults.LateralMovement) {
        Write-Host "`nLateral Movement Detected:" -ForegroundColor Red
        $CorrelationResults.LateralMovement |
            Sort-Object Time |
            Format-Table -AutoSize
    }

    # Show privilege escalation
    if ($CorrelationResults.PrivilegeEscalation) {
        Write-Host "`nPrivilege Escalation Events:" -ForegroundColor Red
        $CorrelationResults.PrivilegeEscalation |
            Sort-Object Time |
            Format-Table -AutoSize
    }

    return $CorrelationResults
}
```

Automated Threat Hunting with Event Logs

Machine Learning-Based Anomaly Detection

Implementing statistical analysis for behavioral anomalies:

```
python
# Python script for ML-based event log anomaly detection
import pandas as pd
import numpy as np
from sklearn.ensemble import IsolationForest
from sklearn.preprocessing import StandardScaler
import win32evtlog
import json
from datetime import datetime, timedelta

class EventLogAnomalyDetector:
```

```python
    def __init__(self, server='localhost'):
        self.server = server
        self.scaler = StandardScaler()
        self.model = IsolationForest(contamination=0.1, random_state=42)

    def extract_features(self, events):
        """Extract behavioral features from events"""
        features = []

        for event in events:
            feature_vector = {
                'hour_of_day': event['TimeGenerated'].hour,
                'day_of_week': event['TimeGenerated'].weekday(),
                'event_frequency': 0,  # Will be calculated
                'unique_sources': 0,   # Will be calculated
                'failed_login_ratio': 0,
                'privilege_events': 0,
                'process_creation_rate': 0,
                'network_connections': 0,
                'registry_modifications': 0,
                'service_installations': 0
            }

            # Calculate event-specific features
            if event['EventID'] == 4625:  # Failed login
                feature_vector['failed_login_ratio'] = 1
            elif event['EventID'] == 4672:  # Special privileges
                feature_vector['privilege_events'] = 1
            elif event['EventID'] == 4688:  # Process creation
                feature_vector['process_creation_rate'] = 1
            elif event['EventID'] == 3:  # Network connection (Sysmon)
                feature_vector['network_connections'] = 1
            elif event['EventID'] == 13:  # Registry modification
(Sysmon)
                feature_vector['registry_modifications'] = 1
            elif event['EventID'] == 7045:  # Service installation
                feature_vector['service_installations'] = 1

            features.append(feature_vector)

        return pd.DataFrame(features)

    def train_model(self, training_data):
        """Train the anomaly detection model"""
        X = self.scaler.fit_transform(training_data)
        self.model.fit(X)
    def detect_anomalies(self, events):
        """Detect anomalous patterns in events"""
        features = self.extract_features(events)
        X = self.scaler.transform(features)

        # Predict anomalies (-1 for anomaly, 1 for normal)
        predictions = self.model.predict(X)
        anomaly_scores = self.model.score_samples(X)

        # Identify anomalous events
        anomalies = []
        for i, (pred, score) in enumerate(zip(predictions,
anomaly_scores)):
            if pred == -1:
                anomalies.append({
                    'event': events[i],
                    'anomaly_score': score,
                    'features': features.iloc[i].to_dict()
                })

        return anomalies

    def generate_threat_report(self, anomalies):
        """Generate threat hunting report from anomalies"""
        report = {
            'timestamp': datetime.now().isoformat(),
            'total_anomalies': len(anomalies),
            'threat_categories': {},
            'high_risk_events': [],
            'recommendations': []
        }

        # Categorize threats
        for anomaly in anomalies:
            event = anomaly['event']
            score = anomaly['anomaly_score']

            # Determine threat category
            if event['EventID'] in [4624, 4625, 4648]:
                category = 'Authentication Anomaly'
            elif event['EventID'] in [4672, 4673]:
                category = 'Privilege Escalation'
            elif event['EventID'] in [7045, 4697]:
                category = 'Persistence Mechanism'
            elif event['EventID'] in [4688, 4689]:
                category = 'Process Execution Anomaly'
            else:
                category = 'Uncategorized'

            if category not in report['threat_categories']:
                report['threat_categories'][category] = []

            report['threat_categories'][category].append({
                'event_id': event['EventID'],
                'time': event['TimeGenerated'].isoformat(),
                'score': score,
                'details': event.get('Message', '')[:200]
            })

            # Identify high-risk events
            if score < -0.5:  # Highly anomalous
                report['high_risk_events'].append(anomaly)

        # Generate recommendations
        if report['threat_categories'].get('Authentication Anomaly'):
            report['recommendations'].append(
                "Review authentication patterns for potential brute
force or credential stuffing attacks"
            )
        if report['threat_categories'].get('Privilege Escalation'):
            report['recommendations'].append(
                "Investigate accounts with unusual privilege
assignments"
            )
        if report['threat_categories'].get('Persistence Mechanism'):
            report['recommendations'].append(
                "Check for unauthorized scheduled tasks, services, or
registry modifications"
            )

        return report

# Usage example
detector = EventLogAnomalyDetector()
# Train on normal baseline data
# detector.train_model(baseline_events)
# Detect anomalies in new events
# anomalies = detector.detect_anomalies(current_events)
# report = detector.generate_threat_report(anomalies)
```

Event Log Forensics and Incident Response

Evidence Collection and Preservation

Forensically sound event log collection procedures:

```
powershell
# Forensic event log collection script
function Collect-ForensicEventLogs {
    param(
        [string]$TargetComputer,
        [string]$OutputPath = "C:\Forensics\EventLogs",
        [string]$CaseNumber = (Get-Date -Format "yyyy-MM-dd_HHmmss"),
        [switch]$IncludeMemoryDump
    )

    # Create forensic collection structure
    $CollectionPath = Join-Path $OutputPath "Case_$CaseNumber"
    $MetadataFile = Join-Path $CollectionPath "collection_metadata.json"

    New-Item -ItemType Directory -Path $CollectionPath -Force | Out-Null
    New-Item -ItemType Directory -Path (Join-Path $CollectionPath
"EventLogs") -Force | Out-Null
    New-Item -ItemType Directory -Path (Join-Path $CollectionPath
"Artifacts") -Force | Out-Null

    # Initialize metadata
    $Metadata = @{
        CaseNumber = $CaseNumber
        CollectionStartTime = Get-Date -Format "yyyy-MM-dd HH:mm:ss"
        TargetComputer = $TargetComputer
        Collector = $env:USERNAME
        CollectorComputer = $env:COMPUTERNAME
        EventLogsCollected = @()
        Hash = @{}
    }

    Write-Host "Starting forensic collection from $TargetComputer"
-ForegroundColor Green

    # Define critical logs to collect
    $CriticalLogs = @(
        'Security',
        'System',
        'Application',
        'Microsoft-Windows-Sysmon/Operational',
        'Microsoft-Windows-PowerShell/Operational',

'Microsoft-Windows-TerminalServices-LocalSessionManager/Operational',

'Microsoft-Windows-TerminalServices-RemoteConnectionManager/Operational'
,
        'Microsoft-Windows-TaskScheduler/Operational',
        'Microsoft-Windows-Windows Defender/Operational',
        'Microsoft-Windows-Windows Firewall With Advanced
Security/Firewall',
        'Microsoft-Windows-WMI-Activity/Operational'
    )

    # Collect each log file
    foreach ($LogName in $CriticalLogs) {
        try {
            Write-Host "Collecting $LogName..." -ForegroundColor Yellow

            $LogFileName = $LogName -replace '[\\/]', '_'
            $ExportPath = Join-Path (Join-Path $CollectionPath
"EventLogs") "$LogFileName.evtx"

            # Use wevtutil for remote collection
            $Command = "wevtutil epl `"$LogName`" `"$ExportPath`"
/r:$TargetComputer"
            Invoke-Expression $Command

            if (Test-Path $ExportPath) {
                # Calculate hash for integrity
                $Hash = Get-FileHash -Path $ExportPath -Algorithm SHA256

                $Metadata.EventLogsCollected += @{
                    LogName = $LogName
                    FileName = "$LogFileName.evtx"
                    FileSize = (Get-Item $ExportPath).Length
                    SHA256 = $Hash.Hash
                    CollectionTime = Get-Date -Format "yyyy-MM-dd
HH:mm:ss"
                }

                Write-Host "  Successfully collected. SHA256:
$($Hash.Hash.Substring(0,16))..." -ForegroundColor Green
            }
        }
        catch {
            Write-Warning "Failed to collect $LogName : $_"
            $Metadata.EventLogsCollected += @{
                LogName = $LogName
                Error = $_.Exception.Message
            }
        }
    }

    # Collect additional artifacts
    Write-Host "`nCollecting additional artifacts..." -ForegroundColor
Cyan

    # Current running processes
    $Processes = Get-WmiObject Win32_Process -ComputerName
$TargetComputer |
        Select-Object Name, ProcessId, ParentProcessId, CommandLine,
CreationDate
    $Processes | Export-Csv -Path (Join-Path (Join-Path $CollectionPath
"Artifacts") "processes.csv") -NoTypeInformation

    # Network connections
    $NetworkConnections = Get-NetTCPConnection -State Established |
        Select-Object LocalAddress, LocalPort, RemoteAddress,
RemotePort, State, OwningProcess
    $NetworkConnections | Export-Csv -Path (Join-Path (Join-Path
$CollectionPath "Artifacts") "network_connections.csv")
-NoTypeInformation

    # Scheduled tasks
    $ScheduledTasks = Get-ScheduledTask | Where-Object {$_.State -ne
'Disabled'} |
        Select-Object TaskName, TaskPath, State, Author, Date
    $ScheduledTasks | Export-Csv -Path (Join-Path (Join-Path
$CollectionPath "Artifacts") "scheduled_tasks.csv") -NoTypeInformation

    # Services
    $Services = Get-WmiObject Win32_Service -ComputerName
$TargetComputer |
        Where-Object {$_.StartMode -eq 'Auto'} |
        Select-Object Name, DisplayName, PathName, StartMode, State,
StartName
    $Services | Export-Csv -Path (Join-Path (Join-Path $CollectionPath
"Artifacts") "services.csv") -NoTypeInformation

    # Complete metadata
    $Metadata.CollectionEndTime = Get-Date -Format "yyyy-MM-dd HH:mm:ss"
    $Metadata | ConvertTo-Json -Depth 3 | Out-File $MetadataFile
    # Generate collection report
    Write-Host "`n=== FORENSIC COLLECTION COMPLETE ===" -ForegroundColor
Green
    Write-Host "Case Number: $CaseNumber"
    Write-Host "Output Path: $CollectionPath"
    Write-Host "Logs Collected: $($Metadata.EventLogsCollected.Count)"
    Write-Host "Metadata File: $MetadataFile"

    return $CollectionPath
}
```

Chain of Custody Documentation

Maintaining forensic integrity throughout the investigation:

```
powershell
# Chain of custody tracking system
function New-ChainOfCustodyRecord {
    param(
        [string]$EvidencePath,
        [string]$CaseNumber,
        [string]$Description,
        [string]$Custodian = $env:USERNAME
    )

    $ChainOfCustody = @{
        CaseNumber = $CaseNumber
        EvidenceID = [Guid]::NewGuid().ToString()
        Description = $Description
        InitialCustodian = $Custodian
        CreatedDate = Get-Date -Format "yyyy-MM-dd HH:mm:ss"
        TransferHistory = @()
        IntegrityChecks = @()
        AccessLog = @()
    }

    # Calculate initial hash
    if (Test-Path $EvidencePath) {
        $InitialHash = Get-FileHash -Path $EvidencePath -Algorithm
SHA256
        $ChainOfCustody.IntegrityChecks += @{
            Timestamp = Get-Date -Format "yyyy-MM-dd HH:mm:ss"
            HashAlgorithm = "SHA256"
            HashValue = $InitialHash.Hash
```

```powershell
            VerifiedBy = $Custodian
            Status = "Initial"
        }
    }

    # Create custody record file
    $CustodyFile = "$EvidencePath.custody.json"
    $ChainOfCustody | ConvertTo-Json -Depth 3 | Out-File $CustodyFile

    # Protect the custody file
    $Acl = Get-Acl $CustodyFile
    $Acl.SetAccessRuleProtection($true, $false)
    $Permission = New-Object
System.Security.AccessControl.FileSystemAccessRule(
        $Custodian, "FullControl", "Allow"
    )
    $Acl.SetAccessRule($Permission)
    Set-Acl -Path $CustodyFile -AclObject $Acl

    Write-Host "Chain of Custody Record Created:" -ForegroundColor Green
    Write-Host "  Evidence ID: $($ChainOfCustody.EvidenceID)"
    Write-Host "  Case Number: $CaseNumber"
    Write-Host "  Custodian: $Custodian"
    Write-Host "  Hash: $($InitialHash.Hash.Substring(0,16))..."

    return $ChainOfCustody
}

# Transfer custody function
function Transfer-EvidenceCustody {
    param(
        [string]$EvidencePath,
        [string]$FromCustodian,
        [string]$ToCustodian,
        [string]$Reason
    )

    $CustodyFile = "$EvidencePath.custody.json"

    if (Test-Path $CustodyFile) {
        $ChainOfCustody = Get-Content $CustodyFile | ConvertFrom-Json

        # Verify integrity before transfer
        $CurrentHash = Get-FileHash -Path $EvidencePath -Algorithm
SHA256
        $LastHash = $ChainOfCustody.IntegrityChecks[-1].HashValue
        if ($CurrentHash.Hash -eq $LastHash) {
            # Record transfer
            $Transfer = @{
                Timestamp = Get-Date -Format "yyyy-MM-dd HH:mm:ss"
                FromCustodian = $FromCustodian
                ToCustodian = $ToCustodian
                Reason = $Reason
                IntegrityVerified = $true
            }

            $ChainOfCustody.TransferHistory += $Transfer

            # Update integrity check
            $ChainOfCustody.IntegrityChecks += @{
                Timestamp = Get-Date -Format "yyyy-MM-dd HH:mm:ss"
                HashAlgorithm = "SHA256"
                HashValue = $CurrentHash.Hash
                VerifiedBy = $ToCustodian
                Status = "Transfer"
            }

            # Save updated record
            $ChainOfCustody | ConvertTo-Json -Depth 3 | Out-File
$CustodyFile

            Write-Host "Custody Transfer Successful" -ForegroundColor
Green
            Write-Host "  From: $FromCustodian"
            Write-Host "  To: $ToCustodian"
            Write-Host "  Integrity: Verified"
        }
        else {
            Write-Error "INTEGRITY CHECK FAILED! Evidence may have been
tampered with."
            return $false
        }
    }
    else {
        Write-Error "Chain of custody record not found for this
evidence."
        return $false
    }

    return $true
}
```

Performance Optimization and Scalability

High-Performance Event Log Querying

Optimized techniques for large-scale event log analysis:

```
powershell
# High-performance parallel event log processing
function Process-EventLogsParallel {
    param(
        [string[]]$Computers,
        [string[]]$LogNames = @('Security', 'System'),
        [int]$MaxThreads = 10,
        [DateTime]$StartTime = (Get-Date).AddDays(-1)
    )

    $RunspacePool = [runspacefactory]::CreateRunspacePool(1,
$MaxThreads)
    $RunspacePool.Open()

    $Jobs = @()

    # Define the script block for parallel execution
    $ScriptBlock = {
        param($Computer, $LogName, $StartTime)

        $Results = @{
            Computer = $Computer
            LogName = $LogName
            TotalEvents = 0
            CriticalEvents = @()
            Errors = @()
        }

        try {
            # Use XML queries for better performance
            $XmlQuery = @"
<QueryList>
  <Query Id="0" Path="$LogName">
    <Select Path="$LogName">

*[System[TimeCreated[@SystemTime>='$($StartTime.ToUniversalTime().ToStri
ng("yyyy-MM-ddTHH:mm:ss.fffZ"))']]
        and
       (System/Level=1 or System/Level=2 or System/Level=3)]
    </Select>
  </Query>
</QueryList>
"@

            $Events = Get-WinEvent -ComputerName $Computer -FilterXml
$XmlQuery -ErrorAction Stop

            $Results.TotalEvents = $Events.Count

            # Process critical events
            foreach ($Event in $Events) {
                if ($Event.Level -le 2) {  # Critical or Error
                    $Results.CriticalEvents += @{
                        Time = $Event.TimeCreated
                        ID = $Event.Id
                        Message = $Event.Message.Substring(0,
[Math]::Min(200, $Event.Message.Length))
                    }
                }
            }
        }
        catch {
            $Results.Errors += $_.Exception.Message
        }

        return $Results
    }

    # Create jobs for each computer/log combination
    foreach ($Computer in $Computers) {
        foreach ($LogName in $LogNames) {
            $PowerShell =
[powershell]::Create().AddScript($ScriptBlock).AddArgument($Computer).Ad
dArgument($LogName).AddArgument($StartTime)
            $PowerShell.RunspacePool = $RunspacePool

            $Jobs += @{
                PowerShell = $PowerShell
                Handle = $PowerShell.BeginInvoke()
                Computer = $Computer
                LogName = $LogName
            }
        }
    }

    # Wait for all jobs to complete
    Write-Host "Processing $($Jobs.Count) log queries in parallel..."
-ForegroundColor Cyan

    $Results = @()
    $Completed = 0

    while ($Jobs.Handle.IsCompleted -contains $false) {
        $CompletedNow = ($Jobs.Handle.IsCompleted -eq $true).Count
        if ($CompletedNow -gt $Completed) {
            Write-Progress -Activity "Processing Event Logs" -Status
"$CompletedNow of $($Jobs.Count) completed" -PercentComplete
(($CompletedNow / $Jobs.Count) * 100)
            $Completed = $CompletedNow
        }
        Start-Sleep -Milliseconds 100
    }

    # Collect results
    foreach ($Job in $Jobs) {
        $JobResult = $Job.PowerShell.EndInvoke($Job.Handle)
        $Results += $JobResult
        $Job.PowerShell.Dispose()
    }

    $RunspacePool.Close()
    $RunspacePool.Dispose()

    # Generate summary report
    $Summary = @{
        TotalComputers = $Computers.Count
        TotalLogs = $LogNames.Count
        TotalEventsProcessed = ($Results | Measure-Object -Property
TotalEvents -Sum).Sum
        CriticalEventCount = ($Results | ForEach-Object {
$_.CriticalEvents.Count } | Measure-Object -Sum).Sum
        ProcessingTime = (Get-Date) - $StartTime
        Errors = $Results | Where-Object { $_.Errors.Count -gt 0 }
    }

    Write-Host "`n=== PARALLEL PROCESSING COMPLETE ===" -ForegroundColor
Green
    Write-Host "Total Events Processed:
$($Summary.TotalEventsProcessed)"
    Write-Host "Critical Events Found: $($Summary.CriticalEventCount)"
    Write-Host "Processing Time: $($Summary.ProcessingTime)"
    return @{
        Results = $Results
        Summary = $Summary
    }
}
```

Security Information and Event Management (SIEM) Integration

Log Forwarding and Centralization

Implementing Windows Event Forwarding (WEF) for centralized monitoring:

```
xml
<!-- Custom WEF subscription for security events -->
<Subscription
xmlns="http://schemas.microsoft.com/2006/03/windows/events/subscription"
>
  <SubscriptionId>Security-Critical-Events</SubscriptionId>
  <SubscriptionType>SourceInitiated</SubscriptionType>
  <Description>Critical security events from domain
systems</Description>
  <Enabled>true</Enabled>
  <Uri>http://schemas.microsoft.com/wbem/wsman/1/windows/EventLog</Uri>
  <ConfigurationMode>Custom</ConfigurationMode>
  <Delivery Mode="Push">
    <Batching>
      <MaxLatencyTime>30000</MaxLatencyTime>
    </Batching>
    <PushSettings>
      <Heartbeat Interval="3600000"/>
    </PushSettings>
  </Delivery>
  <Query>
    <![CDATA[
    <QueryList>
      <Query Id="0">
        <Select Path="Security">
          *[System[(EventID=4624 or EventID=4625 or EventID=4634 or
                   EventID=4648 or EventID=4672 or EventID=4720 or
                   EventID=4728 or EventID=4732 or EventID=4756 or
                   EventID=4697 or EventID=4698 or EventID=4702 or
                   EventID=1102)]]
        </Select>
        <Select Path="System">
          *[System[(EventID=7045 or EventID=7040)]]
        </Select>
        <Select Path="Microsoft-Windows-PowerShell/Operational">
          *[System[(EventID=4103 or EventID=4104)]]
        </Select>
        <Select Path="Microsoft-Windows-Sysmon/Operational">
          *[System[(EventID=1 or EventID=3 or EventID=7 or
                   EventID=8 or EventID=10 or EventID=11 or
                   EventID=12 or EventID=13 or EventID=15)]]
        </Select>
      </Query>
    </QueryList>
    ]]>
  </Query>
  <ReadExistingEvents>true</ReadExistingEvents>
  <TransportName>HTTP</TransportName>
  <ContentFormat>RenderedText</ContentFormat>
  <Locale Language="en-US"/>
  <LogFile>ForwardedEvents</LogFile>
  <PublisherName>Microsoft-Windows-EventCollector</PublisherName>
  <AllowedSourceNonDomainComputers></AllowedSourceNonDomainComputers>

<AllowedSourceDomainComputers>O:NSG:NSD:(A;;GA;;;DC)(A;;GA;;;NS)</Allowe
dSourceDomainComputers>
</Subscription>
```

SIEM Rule Development

Creating detection rules for common attack patterns:

```
yaml
# Splunk detection rule for credential dumping
title: Credential Dumping via LSASS Access
id: credential_dump_lsass_001
status: production
description: Detects suspicious access to LSASS process memory
references:
  - https://attack.mitre.org/techniques/T1003/001/
logsource:
  product: windows
  service: sysmon
detection:
  selection:
    EventID: 10
```

```
     TargetImage|endswith: '\lsass.exe'
     GrantedAccess|contains:
       - '0x1010'
       - '0x1410'
       - '0x1438'
       - '0x143a'
       - '0x1418'
   filter:
     SourceImage|startswith:
       - 'C:\Windows\System32\'
       - 'C:\Windows\SysWOW64\'
     SourceImage|endswith:
       - '\wmiprvse.exe'
       - '\taskmgr.exe'
       - '\procexp.exe'
   condition: selection and not filter
 falsepositives:
   - Legitimate software accessing LSASS
   - Security products
 level: high
 tags:
   - attack.credential_access
   - attack.t1003.001

 ---
 # ElasticSearch detection query
 GET /winlogbeat-*/_search
 {
   "query": {
     "bool": {
       "must": [
         {
           "term": {
             "event.code": "4688"
           }
         },
         {
           "wildcard": {
             "process.command_line": "*sekurlsa*"
           }
         }
       ],
       "filter": {
         "range": {
           "@timestamp": {
             "gte": "now-1h"
```

```
          }
        }
      }
    }
  },
  "aggs": {
    "by_host": {
      "terms": {
        "field": "host.name"
      }
    }
  }
}
```

Compliance and Regulatory Requirements

Event Log Retention Policies

Implementing compliant log retention strategies:

Regulation	Minimum Retention	Event Categories	Special Requirements
PCI DSS	1 year (3 months online)	All security events, access logs	Daily review required
HIPAA	6 years	PHI access, system activity	Encryption required
GDPR	As needed for purpose	Personal data processing	Right to erasure considerations
SOX	7 years	Financial system access	Tamper-proof storage
ISO 27001	3 years minimum	Security incidents, access control	Regular audit trails
NIST 800-53	30-90 days online, 1 year archive	All AU family controls	Automated analysis required

Audit Configuration Scripts

```
powershell
# Configure comprehensive security auditing
function Configure-SecurityAuditing {
    param(
        [string]$ComplianceStandard = "PCI-DSS"
    )

    $AuditSettings = @{
        'PCI-DSS' = @{
            AuditPolicies = @(
                "AuditLogonEvents=3",
                "AuditAccountLogon=3",
                "AuditAccountManage=3",
                "AuditProcessTracking=3",
                "AuditDSAccess=3",
                "AuditPrivilegeUse=3",
                "AuditSystemEvents=3",
                "AuditObjectAccess=3",
                "AuditPolicyChange=3"
            )
            AdvancedAudit = @{
                "Logon/Logoff" = @{
                    "Logon" = "Success,Failure"
                    "Logoff" = "Success"
                    "Account Lockout" = "Success,Failure"
                    "Special Logon" = "Success"
                }
                "Object Access" = @{
                    "File System" = "Success,Failure"
                    "Registry" = "Success,Failure"
                    "SAM" = "Success,Failure"
                }
                "Privilege Use" = @{
                    "Sensitive Privilege Use" = "Success,Failure"
                }
            }
        }
        'HIPAA' = @{
            # HIPAA-specific settings
        }
        'SOX' = @{
            # SOX-specific settings
        }
    }

    # Apply audit policies
    $Settings = $AuditSettings[$ComplianceStandard]

    Write-Host "Configuring audit policies for $ComplianceStandard
compliance..." -ForegroundColor Cyan
```

```
    # Configure basic audit policies
    foreach ($Policy in $Settings.AuditPolicies) {
        auditpol /set /category:$Policy
    }

    # Configure advanced audit policies
    foreach ($Category in $Settings.AdvancedAudit.Keys) {
        foreach ($Subcategory in
$Settings.AdvancedAudit[$Category].Keys) {
            $Setting = $Settings.AdvancedAudit[$Category][$Subcategory]
            auditpol /set /subcategory:"$Subcategory" /success:enable
/failure:enable
        }
    }

    # Configure log sizes
    wevtutil sl Security /ms:4194240  # 4GB for Security log
    wevtutil sl System /ms:1073741824  # 1GB for System log
    wevtutil sl Application /ms:1073741824  # 1GB for Application log

    # Enable command line auditing
    $RegPath =
"HKLM:\Software\Microsoft\Windows\CurrentVersion\Policies\System\Audit"
    New-Item -Path $RegPath -Force | Out-Null
    Set-ItemProperty -Path $RegPath -Name
"ProcessCreationIncludeCmdLine_Enabled" -Value 1

    Write-Host "Audit configuration complete for $ComplianceStandard"
-ForegroundColor Green
}
```

Best Practices and Recommendations

Security Event Log Monitoring Checklist

Critical monitoring requirements for enterprise environments:

✅ **Authentication Monitoring**

- Monitor all logon types (Interactive, Network, Service, RemoteInteractive)
- Track failed authentication attempts (threshold: 5 within 5 minutes)
- Alert on authentication from unusual locations or times
- Monitor service account usage patterns

✅ **Privilege Escalation Detection**

- Track special privilege assignments (Event ID 4672)
- Monitor group membership changes (especially Domain Admins)
- Alert on unusual use of administrative accounts
- Detect token manipulation attempts

✅ **Lateral Movement Indicators**

- Monitor network logons across systems
- Track explicit credential usage (Event ID 4648)
- Detect PsExec and similar tool usage
- Watch for unusual service installations

✅ **Persistence Mechanism Detection**

- Monitor scheduled task creation/modification
- Track new service installations
- Watch registry Run key modifications
- Alert on WMI event subscription creation

✅ **Data Exfiltration Signs**

- Monitor large file access patterns
- Track removable media usage
- Watch for unusual network connections
- Detect cloud storage application usage

Log Volume Estimation and Sizing

Planning storage requirements for event logs:

Environment Size	Daily Log Volume	30-Day Storage	1-Year Archive	Recommended SIEM
Small (< 100 endpoints)	5-10 GB	300 GB	3.6 TB	Splunk Free/ELK
Medium (100-1000)	50-100 GB	3 TB	36 TB	Splunk Enterprise
Large (1000-10000)	500 GB - 1 TB	30 TB	365 TB	Splunk/QRadar
Enterprise (10000+)	5+ TB	150+ TB	1.8+ PB	Splunk/Sentinel

Performance Tuning Recommendations

```
powershell
# Optimize event log performance
function Optimize-EventLogPerformance {
    # Increase event log service thread pool
    Set-ItemProperty -Path
"HKLM:\SYSTEM\CurrentControlSet\Services\EventLog" `
                     -Name "ServiceDllThreadMax" -Value 10

    # Configure channel settings for performance
    $Channels = @(
        "Microsoft-Windows-Security-Auditing",
        "Microsoft-Windows-Sysmon/Operational",
        "Microsoft-Windows-PowerShell/Operational"
    )

    foreach ($Channel in $Channels) {
        # Increase channel reliability
        wevtutil sl $Channel /rt:true /ab:true

        # Set retention policy
        wevtutil sl $Channel /rt:false /ab:false /ms:4294967296  # 4GB
    }

    # Configure WEF for optimal performance
    Set-ItemProperty -Path
"HKLM:\SOFTWARE\Policies\Microsoft\Windows\EventLog\EventForwarding\Subs
criptionManager" `
                     -Name "1" -Value
"Server=http://COLLECTOR:5985/wsman/SubscriptionManager/WEC"

    # Enable high-performance counters
    logman start "Event Log Performance" -p "Microsoft-Windows-EventLog"
0xffffffffffffffff 0xff -ets

    Write-Host "Event log performance optimizations applied"
-ForegroundColor Green
}
```

Related Articles and Resources

- Microsoft Windows Security Auditing Documentation
- MITRE ATT&CK - Windows Event Log Analysis
- NSA Windows Event Monitoring Guidance
- SANS Windows Event Log Cheat Sheet
- Palantir Windows Event Forwarding Guidance
- ACSC Windows Event Logging and Forwarding
- Elastic Windows Event Log Module
- Splunk Windows Event Log Best Practices
- CrowdStrike Falcon Event Search
- FireEye Windows Event Log Analysis
- Digital Forensics - Windows Event Log Analysis
- Incident Response - Event Log Timeline Analysis
- NIST Cybersecurity Framework - Logging Guidance
- Windows Security Log Encyclopedia
- EventID.Net - Windows Event ID Database