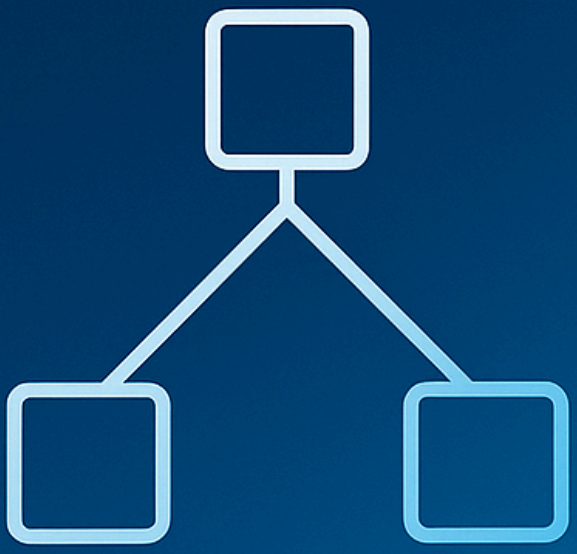
ACTIVE DIRECTORY SECURITY

Active Directory Security Event Monitoring

Enterprise Domain Protection & Threat Detection



Okan YILDIZ • 2025 October

Active Directory Security Event Monitoring: Enterprise Domain Protection and Threat Detection

Executive Summary

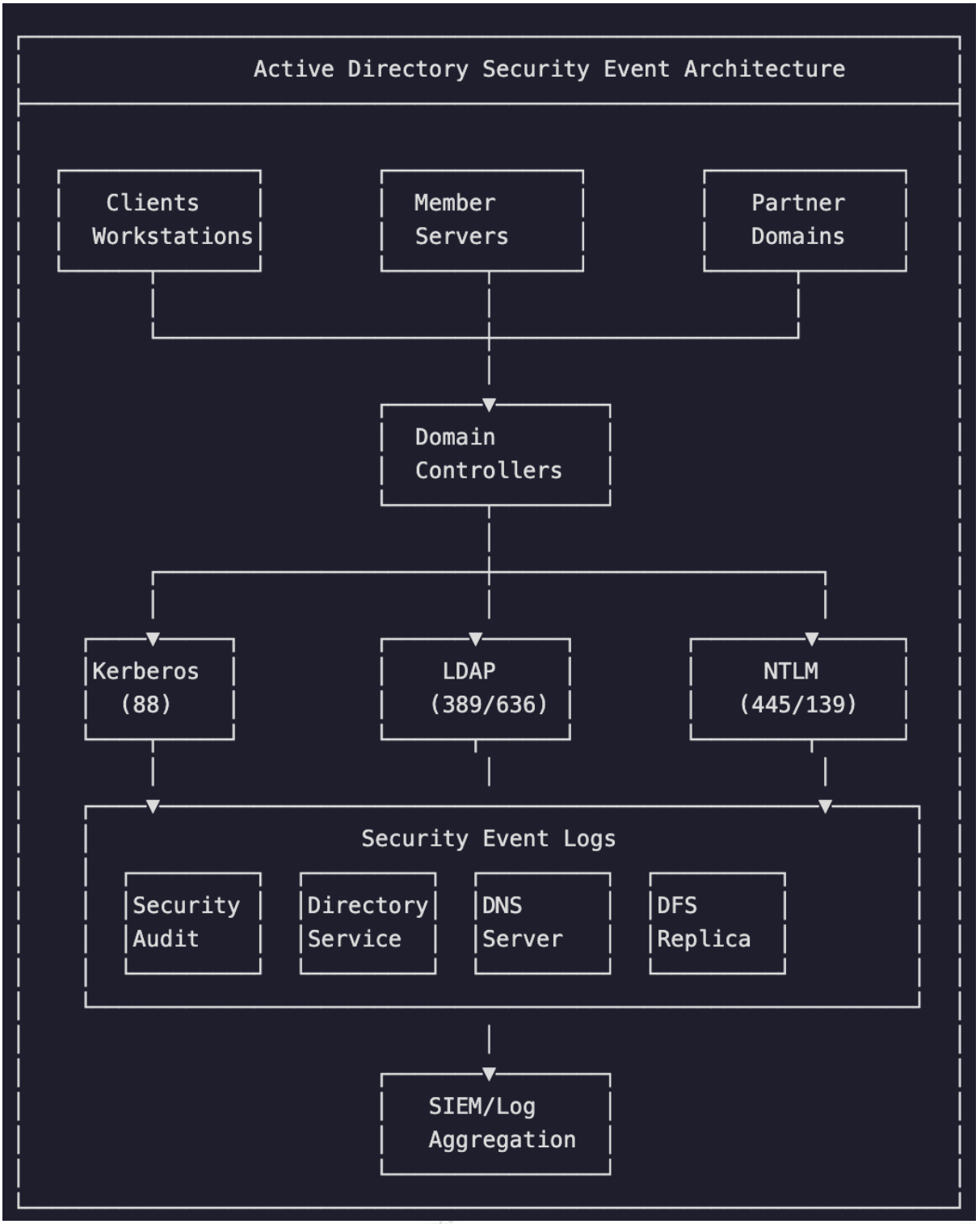
Active Directory (AD) serves as the identity backbone for 90% of Fortune 1000 companies, making it the **crown jewel target for advanced persistent threats (APTs)**. A single compromise of AD can provide attackers with complete control over an entire enterprise network, making robust security monitoring not just important, but absolutely critical for organizational survival.

This comprehensive guide explores advanced Active Directory security event monitoring techniques, from understanding the underlying authentication protocols to detecting sophisticated attacks like Golden Ticket, DCSync, and Kerberoasting. We'll implement real-world detection strategies that security teams need to identify and respond to the most dangerous threats targeting enterprise domains.

Understanding Active Directory Security Architecture

Domain Controller Security Event Flow

Active Directory's security event architecture involves complex interactions between multiple components:



Critical AD Security Event Categories

Domain Controller event logs requiring monitoring:

Log Source	Event Types	Security Relevance	Default Location
Security Log	Authentication, Authorization	User logons, privilege changes	`%SystemRoot%\System32\Winevt\Logs\Security.evtx`
Directory Service	AD replication, LDAP queries	Database modifications, replication issues	`%SystemRoot%\System32\Winevt\Logs\Directory Service.evtx`
DNS Server	DNS queries, zone transfers	DNS poisoning, reconnaissance	`%SystemRoot%\System32\Winevt\Logs\DNS Server.evtx`
Key Management Service	Certificate operations	Certificate abuse, PKI attacks	`%SystemRoot%\System32\Winevt\Logs\Key Management Service.evtx`
AD Web Services	PowerShell AD cmdlets	Remote AD enumeration	`Applications and Services Logs\Microsoft\Windows\Active Directory Web Services`
NTFRS/DFSR	SYSVOL replication	GPO tampering	`%SystemRoot%\System32\Winevt\Logs\File Replication Service.evtx`

Critical Active Directory Security Events

Authentication and Kerberos Events

Understanding Kerberos authentication flow is essential for threat detection:

```
powershell
# Advanced Kerberos monitoring script
function Monitor-KerberosAuthentication {
    param(
        [string[]]$DomainControllers = (Get-ADDomainController -Filter
*).HostName,
        [int]$Hours = 24,
```

```
        [switch]$DetectAnomalies
    )

    $StartTime = (Get-Date).AddHours(-$Hours)
    $KerberosEvents = @{
        # TGT Operations
        4768 = @{
            Name = "Kerberos TGT Request"
            Severity = "Info"
            Fields = @{
                Account = 0
                SourceIP = 9
                ErrorCode = 6
            }
        }
        4771 = @{
            Name = "Kerberos Pre-Auth Failed"
            Severity = "Warning"
            Fields = @{
                Account = 0
                SourceIP = 6
                FailureCode = 4
            }
        }
        4770 = @{
            Name = "Kerberos TGT Renewed"
            Severity = "Info"
            Fields = @{
                Account = 0
                SourceIP = 9
            }
        }

        # Service Ticket Operations
        4769 = @{
            Name = "Kerberos Service Ticket Request"
            Severity = "Info"
            Fields = @{
                Account = 0
                Service = 2
                SourceIP = 10
                TicketOptions = 6
                EncryptionType = 7
            }
        }
        4773 = @{
```

```
            Name = "Kerberos Service Ticket Failed"
            Severity = "Warning"
            Fields = @{
                Account = 0
                Service = 2
                FailureCode = 4
            }
        }
    }

    $Results = @{
        TotalEvents = 0
        Anomalies = @()
        SuspiciousPatterns = @()
        Statistics = @{}
    }

    foreach ($DC in $DomainControllers) {
        Write-Host "Analyzing Domain Controller: $DC" -ForegroundColor
Cyan

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

                if ($Events) {
                    $Results.TotalEvents += $Events.Count
                    $Results.Statistics[$EventID] = $Events.Count

                    # Detect anomalies
                    if ($DetectAnomalies) {
                        # Check for RC4 encryption (potential downgrade
attack)
                        if ($EventID -eq 4769) {
                            $RC4Usage = $Events | Where-Object {
                                $_.Properties[7].Value -eq 0x17  #
RC4-HMAC
                            }
                            if ($RC4Usage) {
                                $Results.Anomalies += [PSCustomObject]@{
                                    Type = "RC4 Encryption Detected"
```

```
                                    Severity = "High"
                                    Count = $RC4Usage.Count
                                    DC = $DC
                                    Accounts = ($RC4Usage |
ForEach-Object {
                                        $_.Properties[0].Value
                                    } | Select-Object -Unique)
                                }
                            }
                        }

                        # Check for TGT requests from non-standard hours
                        if ($EventID -eq 4768) {
                            $AfterHours = $Events | Where-Object {
                                $Hour = $_.TimeCreated.Hour
                                $Hour -lt 6 -or $Hour -gt 22
                            }
                            if ($AfterHours.Count -gt 10) {
                                $Results.SuspiciousPatterns +=
[PSCustomObject]@{
                                    Pattern = "Unusual Hour
Authentication"
                                    Count = $AfterHours.Count
                                    DC = $DC
                                    TimeRange = "Outside 6AM-10PM"
                                }
                            }
                        }

                        # Check for failed pre-authentication (password
spraying)
                        if ($EventID -eq 4771) {
                            $FailedAuth = $Events | Group-Object
-Property {
                                $_.Properties[6].Value  # Source IP
                            } | Where-Object { $_.Count -gt 10 }

                            if ($FailedAuth) {
                                $Results.Anomalies += [PSCustomObject]@{
                                    Type = "Potential Password Spray"
                                    Severity = "Critical"
                                    DC = $DC
                                    SourceIPs = $FailedAuth.Name
                                    AttemptCounts = $FailedAuth.Count
                                }
                            }
```

```
                    }
                }
            }
        }
        catch {
            Write-Warning "Error processing $EventID on $DC : $_"
        }
    }
}

    # Generate report
    Write-Host "`n=== KERBEROS MONITORING REPORT ===" -ForegroundColor
Yellow
    Write-Host "Total Kerberos Events: $($Results.TotalEvents)"

    if ($Results.Anomalies) {
        Write-Host "`nSECURITY ANOMALIES DETECTED:" -ForegroundColor Red
        $Results.Anomalies | Format-Table -AutoSize
    }

    if ($Results.SuspiciousPatterns) {
        Write-Host "`nSUSPICIOUS PATTERNS:" -ForegroundColor Magenta
        $Results.SuspiciousPatterns | Format-Table -AutoSize
    }

    return $Results
}
```

Privileged Group Monitoring

Critical groups requiring continuous monitoring:

Group	Security Impact	Monitor Event IDs	Alert Priority
Domain Admins	Full domain control	4728, 4729, 4756, 4757	**CRITICAL**
Enterprise Admins	Forest-wide control	4728, 4729, 4756, 4757	**CRITICAL**
Schema Admins	Schema modifications	4728, 4729, 4756, 4757	**CRITICAL**
Administrators	DC local admin	4732, 4733	**HIGH**

Account Operators	User/group management	4728, 4729, 4732, 4733	**HIGH**
Backup Operators	File system access	4732, 4733	**MEDIUM**
DNSAdmins	DNS service control	4732, 4733	**HIGH**
Group Policy Creator Owners	GPO creation	4728, 4729	**HIGH**

Advanced Attack Detection Patterns

Golden Ticket Detection

Detecting Golden Ticket attacks through behavioral analysis:

```
powershell
# Golden Ticket detection framework
function Detect-GoldenTicket {
    param(
        [string[]]$DomainControllers = (Get-ADDomainController -Filter
*).HostName,
        [int]$MaxTicketLifetimeHours = 10,
        [switch]$EnableRealTimeAlerts
    )

    $Indicators = @{
        GoldenTicket = @()
        SilverTicket = @()
        TicketAnomalies = @()
    }

    foreach ($DC in $DomainControllers) {
        Write-Host "Scanning $DC for ticket anomalies..."
-ForegroundColor Yellow

        # Event 4769 - Service ticket operations
        $ServiceTickets = Get-WinEvent -ComputerName $DC
-FilterHashtable @{
            LogName = 'Security'
            ID = 4769
            StartTime = (Get-Date).AddHours(-24)
        } -ErrorAction SilentlyContinue

        foreach ($Event in $ServiceTickets) {
```

```
            $EventXML = [xml]$Event.ToXml()

            # Extract ticket properties
            $TicketInfo = @{
                Account = $Event.Properties[0].Value
                Domain = $Event.Properties[1].Value
                ServiceName = $Event.Properties[2].Value
                ServiceID = $Event.Properties[3].Value
                TicketOptions = $Event.Properties[6].Value
                EncryptionType = $Event.Properties[7].Value
                ClientAddress = $Event.Properties[10].Value
                TimeCreated = $Event.TimeCreated
            }

            # Indicator 1: Service ticket without prior TGT request
            $TGTSearch = Get-WinEvent -ComputerName $DC -FilterHashtable
@{
                LogName = 'Security'
                ID = 4768
                StartTime = $TicketInfo.TimeCreated.AddMinutes(-5)
                EndTime = $TicketInfo.TimeCreated
            } -ErrorAction SilentlyContinue |
            Where-Object { $_.Properties[0].Value -eq
$TicketInfo.Account }

            if (-not $TGTSearch) {
                $Indicators.GoldenTicket += [PSCustomObject]@{
                    Indicator = "Service Ticket without TGT"
                    Account = $TicketInfo.Account
                    Service = $TicketInfo.ServiceName
                    Time = $TicketInfo.TimeCreated
                    DC = $DC
                    Severity = "CRITICAL"
                }
            }

            # Indicator 2: Ticket with unusual encryption type for
krbtgt
            if ($TicketInfo.ServiceName -eq 'krbtgt' -and
                $TicketInfo.EncryptionType -notin @(0x12, 0x17)) {  #
AES256, AES128
                $Indicators.GoldenTicket += [PSCustomObject]@{
                    Indicator = "Unusual krbtgt Encryption"
                    Account = $TicketInfo.Account
                    EncType = "0x{0:X}" -f $TicketInfo.EncryptionType
                    Time = $TicketInfo.TimeCreated
```

```
                    DC = $DC
                    Severity = "HIGH"
                }
            }

            # Indicator 3: Ticket lifetime anomaly
            if ($TicketInfo.ServiceName -eq 'krbtgt') {
                # Check for tickets with lifetime > default (10 hours)
                $RenewalEvents = Get-WinEvent -ComputerName $DC
-FilterHashtable @{
                    LogName = 'Security'
                    ID = 4770
                    StartTime = $TicketInfo.TimeCreated
                } -ErrorAction SilentlyContinue |
                Where-Object {
                    $_.Properties[0].Value -eq $TicketInfo.Account -and
                    ($_.TimeCreated -
$TicketInfo.TimeCreated).TotalHours -gt $MaxTicketLifetimeHours
                }

                if ($RenewalEvents) {
                    $Indicators.TicketAnomalies += [PSCustomObject]@{
                        Indicator = "Extended Ticket Lifetime"
                        Account = $TicketInfo.Account
                        LifetimeHours =
[math]::Round(($RenewalEvents[0].TimeCreated -
$TicketInfo.TimeCreated).TotalHours, 2)
                        Time = $TicketInfo.TimeCreated
                        DC = $DC
                        Severity = "MEDIUM"
                    }
                }
            }
        }

        # Event 4624 - Logon events with suspicious ticket use
        $LogonEvents = Get-WinEvent -ComputerName $DC -FilterHashtable
@{
            LogName = 'Security'
            ID = 4624
            StartTime = (Get-Date).AddHours(-24)
        } -ErrorAction SilentlyContinue |
        Where-Object { $_.Properties[8].Value -eq 3 }  # Network logon

        foreach ($Logon in $LogonEvents) {
            # Check for logons with administratively created tickets
```

```
            if ($Logon.Properties[5].Value -match "krbtgt" -or
                $Logon.Properties[11].Value -eq "-") {  # No workstation
name

                $Indicators.GoldenTicket += [PSCustomObject]@{
                    Indicator = "Suspicious Network Logon"
                    Account = $Logon.Properties[5].Value
                    SourceIP = $Logon.Properties[18].Value
                    Time = $Logon.TimeCreated
                    DC = $DC
                    Severity = "HIGH"
                }
            }
        }
    }

    # Alert on critical findings
    if ($EnableRealTimeAlerts -and $Indicators.GoldenTicket) {
        Send-SecurityAlert -Type "Golden Ticket Attack" -Details
$Indicators.GoldenTicket
    }

    # Generate detection report
    Write-Host "`n=== GOLDEN TICKET DETECTION REPORT ==="
-ForegroundColor Red

    if ($Indicators.GoldenTicket) {
        Write-Host "`nGOLDEN TICKET INDICATORS DETECTED:"
-ForegroundColor Red
        $Indicators.GoldenTicket | Format-Table -AutoSize
    }

    if ($Indicators.TicketAnomalies) {
        Write-Host "`nTICKET ANOMALIES:" -ForegroundColor Yellow
        $Indicators.TicketAnomalies | Format-Table -AutoSize
    }

    return $Indicators
}

# Helper function for security alerts
function Send-SecurityAlert {
    param($Type, $Details)
    # Implement your alert mechanism here (email, SIEM, webhook, etc.)
    Write-Host "SECURITY ALERT: $Type" -ForegroundColor Red
-BackgroundColor Yellow
```

```
}
```

DCSync Attack Detection

Monitoring for replication abuse and DCSync attacks:

```
powershell
# DCSync detection and monitoring
function Detect-DCSyncAttack {
    param(
        [string[]]$DomainControllers = (Get-ADDomainController -Filter
*).HostName,
        [switch]$CheckHistoricalData,
        [int]$DaysBack = 7
    )

    $DCReplicationRights = @{
        "DS-Replication-Get-Changes" =
"1131f6aa-9c07-11d1-f79f-00c04fc2dcd2"
        "DS-Replication-Get-Changes-All" =
"1131f6ad-9c07-11d1-f79f-00c04fc2dcd2"
        "DS-Replication-Get-Changes-In-Filtered-Set" =
"89e95b76-444d-4c62-991a-0facbeda640c"
    }

    $DetectedThreats = @{
        DCSync = @()
        UnauthorizedReplication = @()
        PermissionChanges = @()
    }

    foreach ($DC in $DomainControllers) {
        Write-Host "Analyzing replication events on $DC..."
-ForegroundColor Cyan

        # Event 4662 - Directory Service Access
        $DSAccessEvents = Get-WinEvent -ComputerName $DC
-FilterHashtable @{
            LogName = 'Security'
            ID = 4662
            StartTime = if ($CheckHistoricalData) {
                (Get-Date).AddDays(-$DaysBack)
            } else {
                (Get-Date).AddHours(-4)
            }
```

```
        } -ErrorAction SilentlyContinue

        foreach ($Event in $DSAccessEvents) {
            $EventData = @{
                Account = $Event.Properties[1].Value
                ObjectDN = $Event.Properties[6].Value
                AccessMask = $Event.Properties[9].Value
                Properties = $Event.Properties[10].Value
                Time = $Event.TimeCreated
            }

            # Check for replication rights usage
            foreach ($Right in $DCReplicationRights.Values) {
                if ($EventData.Properties -match $Right) {
                    # Verify if account is authorized DC
                    $AuthorizedDCs = (Get-ADDomainController -Filter
*).ComputerObjectDN

                    if ($EventData.Account -notin $AuthorizedDCs -and
                        $EventData.Account -notmatch "SYSTEM|LOCAL
SERVICE") {

                        $DetectedThreats.DCSync += [PSCustomObject]@{
                            Type = "DCSync Attack Detected"
                            Account = $EventData.Account
                            TargetDC = $DC
                            ObjectAccessed = $EventData.ObjectDN
                            Time = $EventData.Time
                            ReplicationRight =
($DCReplicationRights.GetEnumerator() |
                                Where-Object { $_.Value -eq $Right
}).Name
                        }
                    }
                }
            }
        }

        # Event 4929 - Active Directory replica source naming context
removed
        $ReplicationEvents = Get-WinEvent -ComputerName $DC
-FilterHashtable @{
            LogName = 'Security'
            ID = @(4928, 4929, 4930, 4931, 4932, 4933, 4934, 4935, 4936,
4937)
            StartTime = (Get-Date).AddHours(-4)
```

```
        } -ErrorAction SilentlyContinue

        foreach ($ReplEvent in $ReplicationEvents) {
            $SourceDC = $ReplEvent.Properties[3].Value
            $DestDC = $ReplEvent.Properties[4].Value

            # Check for non-DC replication attempts
            if ($SourceDC -notmatch "CN=NTDS Settings" -or
                $DestDC -notmatch "CN=NTDS Settings") {

                $DetectedThreats.UnauthorizedReplication +=
[PSCustomObject]@{
                    EventID = $ReplEvent.Id
                    Description = Switch ($ReplEvent.Id) {
                        4928 { "AD Replica Source Naming Context
Established" }
                        4929 { "AD Replica Source Naming Context
Removed" }
                        4932 { "Synchronization of AD Naming Context
Started" }
                        4933 { "Synchronization of AD Naming Context
Failed" }
                        default { "AD Replication Event" }
                    }
                    Source = $SourceDC
                    Destination = $DestDC
                    Time = $ReplEvent.TimeCreated
                    DC = $DC
                }
            }
        }
    }

    # Check for permission changes that enable DCSync
    $PermissionAudits = Get-WinEvent -FilterHashtable @{
        LogName = 'Security'
        ID = 5136  # Directory Service Changes
        StartTime = (Get-Date).AddHours(-24)
    } -ErrorAction SilentlyContinue |
    Where-Object {
        $_.Message -match
"1131f6aa-9c07-11d1-f79f-00c04fc2dcd2|1131f6ad-9c07-11d1-f79f-00c04fc2dc
d2"
    }

    foreach ($Audit in $PermissionAudits) {
```

```
        $DetectedThreats.PermissionChanges += [PSCustomObject]@{
            Type = "Replication Rights Granted"
            ModifiedBy = ($Audit.Message | Select-String "Account
Name:\s+(\S+)" -AllMatches).Matches[1].Groups[1].Value
            TargetObject = ($Audit.Message | Select-String "Object
DN:\s+(.+)" -AllMatches).Matches[0].Groups[1].Value
            Time = $Audit.TimeCreated
        }
    }

    # Generate report
    Write-Host "`n=== DCSYNC DETECTION REPORT ===" -ForegroundColor Red

    if ($DetectedThreats.DCSync) {
        Write-Host "`nDCSYNC ATTACKS DETECTED:" -ForegroundColor Red
        $DetectedThreats.DCSync | Format-Table -AutoSize
    }

    if ($DetectedThreats.UnauthorizedReplication) {
        Write-Host "`nUNAUTHORIZED REPLICATION:" -ForegroundColor Yellow
        $DetectedThreats.UnauthorizedReplication | Format-Table
-AutoSize
    }

    if ($DetectedThreats.PermissionChanges) {
        Write-Host "`nPERMISSION CHANGES:" -ForegroundColor Magenta
        $DetectedThreats.PermissionChanges | Format-Table -AutoSize
    }

    return $DetectedThreats
}
```

Kerberoasting Detection

Identifying service account credential harvesting:

```
powershell
# Kerberoasting detection system
function Detect-Kerberoasting {
    param(
        [string[]]$DomainControllers = (Get-ADDomainController -Filter
*).HostName,
        [int]$ThresholdPerHour = 10,
        [switch]$AnalyzeServiceAccounts
    )
```

```
    $KerberoastIndicators = @{
        SuspiciousRequests = @()
        VulnerableAccounts = @()
        AnomalousPatterns = @()
    }

    foreach ($DC in $DomainControllers) {
        Write-Host "Analyzing Kerberoasting indicators on $DC..."
-ForegroundColor Yellow

        # Event 4769 - Service Ticket Requests
        $ServiceTickets = Get-WinEvent -ComputerName $DC
-FilterHashtable @{
            LogName = 'Security'
            ID = 4769
            StartTime = (Get-Date).AddHours(-24)
        } -ErrorAction SilentlyContinue

        # Group by requesting account and time window
        $RequestsByAccount = $ServiceTickets | Group-Object -Property {
            $_.Properties[0].Value  # Account name
        }

        foreach ($AccountGroup in $RequestsByAccount) {
            # Analyze request patterns
            $TimeWindows = $AccountGroup.Group | Group-Object -Property
{
                [Math]::Floor(($_.TimeCreated -
(Get-Date).Date).TotalHours)
            }

            foreach ($Window in $TimeWindows) {
                # Check for high-volume requests (potential
Kerberoasting)
                if ($Window.Count -gt $ThresholdPerHour) {
                    # Get unique services requested
                    $Services = $Window.Group | ForEach-Object {
                        $_.Properties[2].Value  # Service name
                    } | Select-Object -Unique

                    # Check for RC4 encryption requests (easier to
crack)
                    $RC4Requests = $Window.Group | Where-Object {
                        $_.Properties[7].Value -eq 0x17  # RC4-HMAC
                    }
```

```
                    $KerberoastIndicators.SuspiciousRequests +=
[PSCustomObject]@{
                        Account = $AccountGroup.Name
                        RequestCount = $Window.Count
                        TimeWindow = "Hour $($Window.Name)"
                        UniqueServices = $Services.Count
                        RC4Requests = $RC4Requests.Count
                        DC = $DC
                        Severity = if ($Window.Count -gt 50) {
"CRITICAL" }
                                   elseif ($Window.Count -gt 25) { "HIGH"
}
                                   else { "MEDIUM" }
                    }
                }
            }

            # Check for requests to service accounts with weak
encryption
            $ServiceAccountRequests = $AccountGroup.Group | Where-Object
{
                $ServiceName = $_.Properties[2].Value
                $ServiceName -notmatch "krbtgt|HTTP|CIFS|LDAP|HOST" -and
                $ServiceName -match "^[\w\-]+\/[\w\.\-]+(@[\w\.\-]+)?$"
            }

            if ($ServiceAccountRequests.Count -gt 5) {
                $KerberoastIndicators.AnomalousPatterns +=
[PSCustomObject]@{
                    Pattern = "Multiple Service Account Requests"
                    RequestingAccount = $AccountGroup.Name
                    ServiceAccountsTargeted = ($ServiceAccountRequests |
ForEach-Object {
                        $_.Properties[2].Value
                    } | Select-Object -Unique).Count
                    TotalRequests = $ServiceAccountRequests.Count
                    Time = $ServiceAccountRequests[0].TimeCreated
                    DC = $DC
                }
            }
        }
    }

    # Analyze vulnerable service accounts if requested
    if ($AnalyzeServiceAccounts) {
```

```
        Write-Host "Analyzing vulnerable service accounts..."
-ForegroundColor Cyan

        # Get all service accounts with SPNs
        $ServiceAccounts = Get-ADUser -Filter {ServicePrincipalName
-like "*"} -Properties ServicePrincipalName, PasswordLastSet,
LastLogonDate |
            Where-Object { $_.Enabled -eq $true }

        foreach ($Account in $ServiceAccounts) {
            # Check for weak configurations
            $Vulnerabilities = @()

            # Old password
            if ($Account.PasswordLastSet -lt (Get-Date).AddDays(-365)) {
                $Vulnerabilities += "Password older than 365 days"
            }

            # Check if using RC4 encryption
            $UserAccountControl = (Get-ADUser $Account -Properties
msDS-SupportedEncryptionTypes).'msDS-SupportedEncryptionTypes'
            if ($null -eq $UserAccountControl -or ($UserAccountControl
-band 0x4) -eq 0) {
                $Vulnerabilities += "RC4 encryption enabled"
            }

            # High-privilege service account
            $Groups = Get-ADPrincipalGroupMembership $Account |
Select-Object -ExpandProperty Name
            if ($Groups -match "Domain Admins|Enterprise
Admins|Administrators") {
                $Vulnerabilities += "High privilege account"
            }

            if ($Vulnerabilities) {
                $KerberoastIndicators.VulnerableAccounts +=
[PSCustomObject]@{
                    Account = $Account.SamAccountName
                    SPNs = ($Account.ServicePrincipalName -join ", ")
                    Vulnerabilities = ($Vulnerabilities -join "; ")
                    PasswordAge = [Math]::Round(((Get-Date) -
$Account.PasswordLastSet).TotalDays)
                    LastLogon = $Account.LastLogonDate
                }
            }
        }
```

```
    }

    # Generate detection report
    Write-Host "`n=== KERBEROASTING DETECTION REPORT ==="
-ForegroundColor Red

    if ($KerberoastIndicators.SuspiciousRequests) {
        Write-Host "`nSUSPICIOUS SERVICE TICKET REQUESTS:"
-ForegroundColor Red
        $KerberoastIndicators.SuspiciousRequests |
            Sort-Object RequestCount -Descending |
            Format-Table -AutoSize
    }

    if ($KerberoastIndicators.AnomalousPatterns) {
        Write-Host "`nANOMALOUS PATTERNS DETECTED:" -ForegroundColor
Yellow
        $KerberoastIndicators.AnomalousPatterns | Format-Table -AutoSize
    }

    if ($KerberoastIndicators.VulnerableAccounts) {
        Write-Host "`nVULNERABLE SERVICE ACCOUNTS:" -ForegroundColor
Magenta
        $KerberoastIndicators.VulnerableAccounts |
            Sort-Object PasswordAge -Descending |
            Format-Table -AutoSize
    }

    return $KerberoastIndicators
}
```

LDAP and Directory Service Monitoring

LDAP Attack Detection

Monitoring LDAP operations for enumeration and attacks:

```
powershell
# LDAP monitoring and attack detection
function Monitor-LDAPActivity {
    param(
        [string[]]$DomainControllers = (Get-ADDomainController -Filter
*).HostName,
        [int]$Hours = 4,
        [switch]$DetectEnumeration
```

```
    )

    $LDAPAnalysis = @{
        Statistics = @{}
        EnumerationAttempts = @()
        SuspiciousQueries = @()
        PerformanceIssues = @()
    }

    foreach ($DC in $DomainControllers) {
        Write-Host "Analyzing LDAP activity on $DC..." -ForegroundColor
Cyan

        # Directory Service event log
        try {
            $DSEvents = Get-WinEvent -ComputerName $DC -LogName
"Directory Service" -MaxEvents 10000 |
                Where-Object {
                    $_.TimeCreated -gt (Get-Date).AddHours(-$Hours) -and
                    $_.Id -in @(1644, 1139, 2889, 2887, 2886)
                }

            foreach ($Event in $DSEvents) {
                switch ($Event.Id) {
                    1644 {  # LDAP searches
                        if ($Event.Message -match "Client:\s+([^\s]+)")
{
                            $Client = $Matches[1]

                            # Track LDAP search statistics
                            if (-not
$LDAPAnalysis.Statistics.ContainsKey($Client)) {
                                $LDAPAnalysis.Statistics[$Client] = 0
                            }
                            $LDAPAnalysis.Statistics[$Client]++

                            # Detect enumeration patterns
                            if ($DetectEnumeration) {
                                # Check for sensitive attribute queries
                                if ($Event.Message -match
"userPassword|unixUserPassword|unicodePwd|ms-Mcs-AdmPwd") {
                                    $LDAPAnalysis.SuspiciousQueries +=
[PSCustomObject]@{
                                        Type = "Sensitive Attribute
Query"
                                        Client = $Client
```

```
                                    Attribute = $Matches[0]
                                    Time = $Event.TimeCreated
                                    DC = $DC
                                }
                            }

                            # Check for large result sets
(enumeration)
                            if ($Event.Message -match "Entries
Returned:\s+(\d+)" -and [int]$Matches[1] -gt 1000) {
                                $LDAPAnalysis.EnumerationAttempts +=
[PSCustomObject]@{
                                    Type = "Large LDAP Query"
                                    Client = $Client
                                    EntriesReturned = $Matches[1]
                                    Time = $Event.TimeCreated
                                    DC = $DC
                                }
                            }
                        }
                    }
                }

                2889 {  # Unsigned LDAP bind
                    $LDAPAnalysis.SuspiciousQueries +=
[PSCustomObject]@{
                        Type = "Unsigned LDAP Bind"
                        Details = "Potential MITM vulnerability"
                        Message = $Event.Message
                        Time = $Event.TimeCreated
                        DC = $DC
                    }
                }

                2887 {  # LDAP signing requirement bypassed
                    $LDAPAnalysis.SuspiciousQueries +=
[PSCustomObject]@{
                        Type = "LDAP Signing Bypassed"
                        Details = "Security configuration issue"
                        Time = $Event.TimeCreated
                        DC = $DC
                    }
                }

                1139 {  # LDAP performance issue
                    if ($Event.Message -match
```

```
"expensive|inefficient") {
                            $LDAPAnalysis.PerformanceIssues +=
[PSCustomObject]@{
                                Type = "Expensive LDAP Query"
                                Details = $Event.Message.Substring(0,
[Math]::Min(200, $Event.Message.Length))
                                Time = $Event.TimeCreated
                                DC = $DC
                            }
                        }
                    }
                }
            }
        }
        catch {
            Write-Warning "Could not access Directory Service log on
$DC: $_"
        }

        # Check Security log for LDAP-related events
        $SecurityEvents = Get-WinEvent -ComputerName $DC
-FilterHashtable @{
            LogName = 'Security'
            ID = @(4662, 5136, 5137, 5138, 5139, 5141)
            StartTime = (Get-Date).AddHours(-$Hours)
        } -ErrorAction SilentlyContinue

        # Analyze directory service modifications
        $ModificationEvents = $SecurityEvents | Where-Object { $_.Id -in
@(5136, 5137, 5138, 5139, 5141) }

        if ($ModificationEvents) {
            $ModsByUser = $ModificationEvents | Group-Object -Property {
                ($_.Message | Select-String "Account Name:\s+(\S+)"
-AllMatches).Matches[0].Groups[1].Value
            }

            foreach ($UserMods in $ModsByUser) {
                if ($UserMods.Count -gt 50) {
                    $LDAPAnalysis.SuspiciousQueries +=
[PSCustomObject]@{
                        Type = "Bulk Directory Modifications"
                        Account = $UserMods.Name
                        ModificationCount = $UserMods.Count
                        Time = $UserMods.Group[0].TimeCreated
                        DC = $DC
```

```
                    }
                }
            }
        }
    }

    # Generate LDAP monitoring report
    Write-Host "`n=== LDAP ACTIVITY ANALYSIS ===" -ForegroundColor
Yellow

    if ($LDAPAnalysis.Statistics.Count -gt 0) {
        Write-Host "`nLDAP Client Statistics:" -ForegroundColor Cyan
        $LDAPAnalysis.Statistics.GetEnumerator() |
            Sort-Object Value -Descending |
            Select-Object -First 10 |
            ForEach-Object {
                Write-Host "  $($_.Key): $($_.Value) queries"
            }
    }

    if ($LDAPAnalysis.EnumerationAttempts) {
        Write-Host "`nPOTENTIAL ENUMERATION ATTEMPTS:" -ForegroundColor
Red
        $LDAPAnalysis.EnumerationAttempts | Format-Table -AutoSize
    }

    if ($LDAPAnalysis.SuspiciousQueries) {
        Write-Host "`nSUSPICIOUS LDAP QUERIES:" -ForegroundColor Yellow
        $LDAPAnalysis.SuspiciousQueries | Format-Table -AutoSize
    }

    return $LDAPAnalysis
}
```

Group Policy and SYSVOL Monitoring

GPO Modification Detection

Monitoring Group Policy changes for backdoors and persistence:

```
powershell
# GPO security monitoring system
function Monitor-GroupPolicyChanges {
    param(
        [string]$DomainName = (Get-ADDomain).DNSRoot,
```

```
        [int]$Hours = 24,
        [switch]$CheckSYSVOLIntegrity
    )

    $GPOMonitoring = @{
        ModifiedGPOs = @()
        NewGPOs = @()
        DeletedGPOs = @()
        SYSVOLChanges = @()
        SecurityConcerns = @()
    }

    # Get all GPO modification events
    $DomainControllers = (Get-ADDomainController -Filter *).HostName

    foreach ($DC in $DomainControllers) {
        Write-Host "Checking GPO changes on $DC..." -ForegroundColor
Cyan

        # Event 5136 - Directory Service Changes (GPO modifications)
        $GPOEvents = Get-WinEvent -ComputerName $DC -FilterHashtable @{
            LogName = 'Security'
            ID = @(5136, 5137, 5141)
            StartTime = (Get-Date).AddHours(-$Hours)
        } -ErrorAction SilentlyContinue |
        Where-Object { $_.Message -match "CN=Policies,CN=System" }

        foreach ($Event in $GPOEvents) {
            $GPOInfo = @{
                EventType = switch ($Event.Id) {
                    5136 { "GPO Modified" }
                    5137 { "GPO Created" }
                    5141 { "GPO Deleted" }
                }
                ModifiedBy = ($Event.Message | Select-String "Account
Name:\s+(\S+)" -AllMatches).Matches[1].Groups[1].Value
                ObjectDN = ($Event.Message | Select-String "Object
DN:\s+(.+)" -AllMatches).Matches[0].Groups[1].Value
                Time = $Event.TimeCreated
                DC = $DC
            }

            # Extract GPO GUID
            if ($GPOInfo.ObjectDN -match "\{([A-F0-9\-]+)\}") {
                $GPOInfo.GPOID = $Matches[1]
```

```
                # Try to get GPO name
                try {
                    $GPO = Get-GPO -Guid $GPOInfo.GPOID -ErrorAction
SilentlyContinue
                    $GPOInfo.GPOName = $GPO.DisplayName
                }
                catch {
                    $GPOInfo.GPOName = "Unknown (possibly deleted)"
                }
            }

            # Categorize the change
            switch ($Event.Id) {
                5136 { $GPOMonitoring.ModifiedGPOs +=
[PSCustomObject]$GPOInfo }
                5137 { $GPOMonitoring.NewGPOs +=
[PSCustomObject]$GPOInfo }
                5141 { $GPOMonitoring.DeletedGPOs +=
[PSCustomObject]$GPOInfo }
            }

            # Check for suspicious GPO names or patterns
            if ($GPOInfo.GPOName -match "temp|test|debug|backdoor" -or
                $GPOInfo.ModifiedBy -notmatch "Domain Admins|Group
Policy Creator Owners") {
                $GPOMonitoring.SecurityConcerns += [PSCustomObject]@{
                    Concern = "Suspicious GPO Activity"
                    GPOName = $GPOInfo.GPOName
                    ModifiedBy = $GPOInfo.ModifiedBy
                    Action = $GPOInfo.EventType
                    Time = $GPOInfo.Time
                }
            }
        }
    }

    # Check SYSVOL integrity if requested
    if ($CheckSYSVOLIntegrity) {
        Write-Host "Checking SYSVOL integrity..." -ForegroundColor
Yellow

        $SYSVOLPath = "\\$DomainName\SYSVOL\$DomainName\Policies"

        # Get all GPO folders in SYSVOL
        $SYSVOLGPOs = Get-ChildItem -Path $SYSVOLPath -Directory |
            Where-Object { $_.Name -match "\{[A-F0-9\-]+\}" }
```

```
        foreach ($GPOFolder in $SYSVOLGPOs) {
            # Check for suspicious files
            $SuspiciousFiles = Get-ChildItem -Path $GPOFolder.FullName
-Recurse -File |
                Where-Object {
                    $_.Extension -in @('.exe', '.dll', '.ps1', '.bat',
'.cmd', '.vbs', '.js') -or
                    $_.Name -match 'backdoor|malware|payload|shell'
                }

            if ($SuspiciousFiles) {
                foreach ($File in $SuspiciousFiles) {
                    $GPOMonitoring.SecurityConcerns +=
[PSCustomObject]@{
                        Concern = "Suspicious File in GPO"
                        GPOPath = $GPOFolder.Name
                        FileName = $File.Name
                        FilePath = $File.FullName
                        LastModified = $File.LastWriteTime
                    }
                }
            }

            # Check for recent modifications
            $RecentChanges = Get-ChildItem -Path $GPOFolder.FullName
-Recurse |
                Where-Object { $_.LastWriteTime -gt
(Get-Date).AddHours(-$Hours) }

            if ($RecentChanges) {
                $GPOMonitoring.SYSVOLChanges += [PSCustomObject]@{
                    GPO = $GPOFolder.Name
                    ChangedFiles = $RecentChanges.Count
                    LastChange = ($RecentChanges | Sort-Object
LastWriteTime -Descending | Select-Object -First 1).LastWriteTime
                }
            }
        }
    }

    # Generate monitoring report
    Write-Host "`n=== GROUP POLICY MONITORING REPORT ==="
-ForegroundColor Yellow

    if ($GPOMonitoring.NewGPOs) {
```

```
        Write-Host "`nNEW GPOs CREATED:" -ForegroundColor Green
        $GPOMonitoring.NewGPOs | Format-Table -AutoSize
    }

    if ($GPOMonitoring.ModifiedGPOs) {
        Write-Host "`nMODIFIED GPOs:" -ForegroundColor Yellow
        $GPOMonitoring.ModifiedGPOs | Format-Table -AutoSize
    }

    if ($GPOMonitoring.DeletedGPOs) {
        Write-Host "`nDELETED GPOs:" -ForegroundColor Red
        $GPOMonitoring.DeletedGPOs | Format-Table -AutoSize
    }

    if ($GPOMonitoring.SecurityConcerns) {
        Write-Host "`nSECURITY CONCERNS DETECTED:" -ForegroundColor Red
-BackgroundColor Yellow
        $GPOMonitoring.SecurityConcerns | Format-Table -AutoSize
    }

    return $GPOMonitoring
}
```

Automated Threat Hunting and Response

Machine Learning-Based Anomaly Detection

Implementing ML for advanced AD threat detection:

```
python
# Python script for ML-based AD anomaly detection
import pandas as pd
import numpy as np
from sklearn.ensemble import IsolationForest
from sklearn.preprocessing import StandardScaler
from datetime import datetime, timedelta
import win32evtlog
import win32evtlogutil
import json

class ADSecurityAnalyzer:
    def __init__(self, domain_controllers):
        self.dcs = domain_controllers
        self.model = IsolationForest(contamination=0.05,
random_state=42)
```

```
        self.scaler = StandardScaler()
        self.baseline_data = None

    def extract_features(self, events_df):
        """Extract behavioral features from AD events"""
        features = pd.DataFrame()

        # Time-based features
        features['hour'] =
pd.to_datetime(events_df['TimeGenerated']).dt.hour
        features['day_of_week'] =
pd.to_datetime(events_df['TimeGenerated']).dt.dayofweek
        features['is_weekend'] = features['day_of_week'].isin([5,
6]).astype(int)
        features['is_business_hours'] = features['hour'].between(8,
18).astype(int)

        # Authentication features
        auth_events = events_df[events_df['EventID'].isin([4624, 4625,
4768, 4771])]
        features['auth_failure_rate'] = (
            auth_events[auth_events['EventID'].isin([4625,
4771])].shape[0] /
            max(auth_events.shape[0], 1)
        )

        # Kerberos features
        kerb_events = events_df[events_df['EventID'].isin([4768, 4769,
4770, 4771])]
        features['tgt_requests'] = kerb_events[kerb_events['EventID'] ==
4768].shape[0]
        features['service_tickets'] = kerb_events[kerb_events['EventID']
== 4769].shape[0]
        features['ticket_renewals'] = kerb_events[kerb_events['EventID']
== 4770].shape[0]

        # Privilege escalation indicators
        priv_events = events_df[events_df['EventID'].isin([4672, 4673,
4674])]
        features['privilege_events'] = priv_events.shape[0]

        # Group membership changes
        group_events = events_df[events_df['EventID'].isin([4728, 4729,
4732, 4733, 4756, 4757])]
        features['group_changes'] = group_events.shape[0]
```

```
        # Account management
        account_events = events_df[events_df['EventID'].isin([4720,
4722, 4723, 4724, 4725, 4726])]
        features['account_changes'] = account_events.shape[0]

        # Calculate entropy of source IPs
        if 'SourceIP' in events_df.columns:
            ip_counts = events_df['SourceIP'].value_counts()
            ip_probs = ip_counts / ip_counts.sum()
            features['ip_entropy'] = -sum(ip_probs * np.log2(ip_probs +
1e-10))

        # Unique accounts accessing the system
        if 'AccountName' in events_df.columns:
            features['unique_accounts'] =
events_df['AccountName'].nunique()

            # Check for service account anomalies
            service_accounts =
events_df[events_df['AccountName'].str.endswith('$')]
            features['service_account_activity'] =
service_accounts.shape[0]

        return features

    def train_baseline(self, historical_events):
        """Train the anomaly detection model on historical data"""
        print("Training baseline model on historical data...")

        # Extract features from historical data
        features = self.extract_features(historical_events)

        # Scale features
        scaled_features = self.scaler.fit_transform(features.fillna(0))

        # Train isolation forest
        self.model.fit(scaled_features)
        self.baseline_data = features

        print(f"Baseline trained on {len(historical_events)} events")

    def detect_anomalies(self, current_events):
        """Detect anomalies in current events"""
        if self.baseline_data is None:
            raise ValueError("Model not trained. Run train_baseline
first.")
```

```
        # Extract features
        features = self.extract_features(current_events)

        # Scale features
        scaled_features = self.scaler.transform(features.fillna(0))

        # Predict anomalies
        predictions = self.model.predict(scaled_features)
        anomaly_scores = self.model.score_samples(scaled_features)

        # Create anomaly report
        anomalies = []
        for i, (pred, score) in enumerate(zip(predictions,
anomaly_scores)):
            if pred == -1:  # Anomaly detected
                anomaly_detail = {
                    'timestamp': datetime.now().isoformat(),
                    'anomaly_score': float(score),
                    'features': features.iloc[i].to_dict(),
                    'severity': self.calculate_severity(score,
features.iloc[i])
                }
                anomalies.append(anomaly_detail)

        return anomalies

    def calculate_severity(self, score, features):
        """Calculate anomaly severity based on score and features"""
        severity = "LOW"

        # High-risk indicators
        if features.get('privilege_events', 0) > 5:
            severity = "HIGH"
        elif features.get('group_changes', 0) > 3:
            severity = "HIGH"
        elif features.get('auth_failure_rate', 0) > 0.5:
            severity = "MEDIUM"
        elif score < -0.5:
            severity = "CRITICAL"
        elif score < -0.3:
            severity = "HIGH"
        elif score < -0.1:
            severity = "MEDIUM"

        return severity
```

```
    def generate_threat_report(self, anomalies):
        """Generate comprehensive threat report"""
        report = {
            'timestamp': datetime.now().isoformat(),
            'total_anomalies': len(anomalies),
            'critical_count': sum(1 for a in anomalies if a['severity']
== 'CRITICAL'),
            'high_count': sum(1 for a in anomalies if a['severity'] ==
'HIGH'),
            'threat_categories': {},
            'recommendations': []
        }

        # Categorize threats
        for anomaly in anomalies:
            features = anomaly['features']

            # Determine threat category
            if features.get('privilege_events', 0) > 5:
                category = 'Privilege Escalation'
            elif features.get('auth_failure_rate', 0) > 0.5:
                category = 'Brute Force/Password Spray'
            elif features.get('group_changes', 0) > 3:
                category = 'Unauthorized Group Modification'
            elif features.get('service_tickets', 0) > 100:
                category = 'Potential Kerberoasting'
            elif not features.get('is_business_hours', 1):
                category = 'After-Hours Activity'
            else:
                category = 'Uncategorized Anomaly'

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

            report['threat_categories'][category].append({
                'severity': anomaly['severity'],
                'score': anomaly['anomaly_score'],
                'details': anomaly['features']
            })

        # Generate recommendations
        if 'Privilege Escalation' in report['threat_categories']:
            report['recommendations'].append(
                "Review privileged account usage and verify all
privilege assignments"
```

```
            )
        if 'Brute Force/Password Spray' in report['threat_categories']:
            report['recommendations'].append(
                "Investigate source IPs for authentication failures and
consider blocking"
            )
        if 'Potential Kerberoasting' in report['threat_categories']:
            report['recommendations'].append(
                "Review service account configurations and reset
passwords for high-value accounts"
            )

        return report

# Usage example
analyzer = ADSecurityAnalyzer(['DC01', 'DC02'])
# analyzer.train_baseline(historical_events_df)
# anomalies = analyzer.detect_anomalies(current_events_df)
# report = analyzer.generate_threat_report(anomalies)
```


SIEM Integration and Correlation Rules

Splunk Correlation Rules for AD Security

```
conf
# Splunk correlation searches for AD security monitoring

# Golden Ticket Detection
[Golden Ticket - Service Ticket without TGT]
search = index=wineventlog sourcetype=WinEventLog:Security
EventCode=4769
| eval ticket_time=_time
| join type=left Account_Name
    [search index=wineventlog sourcetype=WinEventLog:Security
EventCode=4768 earliest=-5m@m
    | eval tgt_time=_time]
| where isnull(tgt_time)
| table _time, ComputerName, Account_Name, Service_Name, Client_Address
alert.track = 1
alert.severity = 3
cron_schedule = */5 * * * *
dispatch.earliest_time = -5m
dispatch.latest_time = now

# DCSync Attack Detection
```

```
[DCSync - Replication Rights Usage by Non-DC]
search = index=wineventlog sourcetype=WinEventLog:Security
EventCode=4662
| rex field=Properties "%%(?<Properties>\d+)"
| where Properties IN ("1131f6aa-9c07-11d1-f79f-00c04fc2dcd2",
                       "1131f6ad-9c07-11d1-f79f-00c04fc2dcd2",
                       "89e95b76-444d-4c62-991a-0facbeda640c")
| where NOT match(Account_Name, ".*\$$")
| stats count by Account_Name, ComputerName, Object_Name, _time
alert.track = 1
alert.severity = 1
cron_schedule = */15 * * * *

# Kerberoasting Detection
[Kerberoasting - Excessive Service Ticket Requests]
search = index=wineventlog sourcetype=WinEventLog:Security
EventCode=4769
| where Ticket_Encryption_Type=0x17
| bucket _time span=1h
| stats count as ticket_count, dc(Service_Name) as unique_services by
Account_Name, _time
| where ticket_count > 20 AND unique_services > 5
alert.track = 1
alert.severity = 2
cron_schedule = 0 * * * *

# Password Spray Detection
[Password Spray - Multiple Failed Authentications]
search = index=wineventlog sourcetype=WinEventLog:Security
(EventCode=4625 OR EventCode=4771)
| bucket _time span=5m
| stats dc(Account_Name) as unique_accounts, count as failures by
Source_Network_Address, _time
| where unique_accounts > 10 AND failures > 20
alert.track = 1
alert.severity = 2
cron_schedule = */5 * * * *

# Privileged Group Changes
[Critical Group Membership Changes]
search = index=wineventlog sourcetype=WinEventLog:Security EventCode IN
(4728,4729,4732,4733,4756,4757)
| rex field=Group_Name "(?<Critical_Group>Domain Admins|Enterprise
Admins|Schema Admins|Administrators)"
| where isnotnull(Critical_Group)
| table _time, Subject_Account_Name, Member_Name, Group_Name, Operation
```

```
alert.track = 1
alert.severity = 1
cron_schedule = */10 * * * *
```

ElasticSearch Detection Queries

```
json
// Elasticsearch Watcher for AD security monitoring

{
  "trigger": {
    "schedule": {
      "interval": "5m"
    }
  },
  "input": {
    "search": {
      "request": {
        "search_type": "query_then_fetch",
        "indices": ["winlogbeat-*"],
        "body": {
          "query": {
            "bool": {
              "must": [
                {
                  "term": {
                    "event.code": "4769"
                  }
                },
                {
                  "range": {
                    "@timestamp": {
                      "gte": "now-1h"
                    }
                  }
                }
              ]
            }
          },
          "aggs": {
            "by_account": {
              "terms": {
                "field": "user.name",
                "size": 100
              },
              "aggs": {
```

```
                "unique_services": {
                  "cardinality": {
                    "field": "winlog.event_data.ServiceName"
                  }
                }
              }
            }
          }
        }
      }
    },
    "condition": {
      "script": {
        "source": """
          return ctx.payload.aggregations.by_account.buckets.stream()
            .anyMatch(bucket -> bucket.doc_count > 50 &&
bucket.unique_services.value > 10)
        """
      }
    },
    "actions": {
      "send_alert": {
        "webhook": {
          "scheme": "https",
          "host": "security.example.com",
          "port": 443,
          "method": "post",
          "path": "/api/alerts",
          "body": """
          {
            "alert": "Potential Kerberoasting Attack",
            "severity": "HIGH",
            "details": "{{ctx.payload.aggregations.by_account.buckets}}"
          }
          """
        }
      }
    }
}
```

Best Practices and Security Hardening

AD Security Monitoring Checklist

Essential monitoring requirements for enterprise AD environments:

✅ Authentication Monitoring

- Monitor all Kerberos authentication events (4768, 4769, 4770, 4771)
- Track NTLM authentication usage (4776)
- Alert on authentication from unusual locations
- Monitor service account authentication patterns

✅ Privilege Monitoring

- Track all changes to privileged groups
- Monitor AdminSDHolder modifications
- Alert on privilege escalation events
- Audit delegation changes

✅ Replication Monitoring

- Monitor DCSync indicators
- Track replication metadata changes
- Alert on non-DC replication attempts
- Monitor DRS protocol usage

✅ Object Monitoring

- Track critical object modifications
- Monitor schema changes
- Alert on security descriptor changes
- Audit GPO modifications

✅ Service Account Security

- Monitor SPN modifications
- Track Kerberoasting indicators
- Alert on service account privilege changes
- Monitor delegation settings

Security Configuration Scripts

```
powershell
# Comprehensive AD security hardening script
function Harden-ADSecurity {
    param(
        [switch]$EnableAdvancedAuditing,
```

```
        [switch]$ConfigureSysmon,
        [switch]$HardenKerberos
    )

    Write-Host "Starting Active Directory Security Hardening..."
-ForegroundColor Green

    if ($EnableAdvancedAuditing) {
        # Configure advanced audit policies
        Write-Host "Configuring Advanced Audit Policies..."
-ForegroundColor Yellow

        $AuditPolicies = @(
            "Account Logon,Credential Validation,Success,Failure",
            "Account Logon,Kerberos Authentication
Service,Success,Failure",
            "Account Logon,Kerberos Service Ticket
Operations,Success,Failure",
            "Account Management,Security Group
Management,Success,Failure",
            "Account Management,User Account
Management,Success,Failure",
            "DS Access,Directory Service Access,Success,Failure",
            "DS Access,Directory Service Changes,Success,Failure",
            "Logon/Logoff,Logon,Success,Failure",
            "Logon/Logoff,Special Logon,Success,Failure",
            "Object Access,Central Policy Staging,Success,Failure",
            "Policy Change,Authentication Policy
Change,Success,Failure",
            "Policy Change,Authorization Policy Change,Success,Failure",
            "Privilege Use,Sensitive Privilege Use,Success,Failure"
        )

        foreach ($Policy in $AuditPolicies) {
            $Parts = $Policy -split ','
            auditpol /set /subcategory:"$($Parts[1])" /success:enable
/failure:enable
        }

        # Enable command line process 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
    }
```

```
    if ($HardenKerberos) {
        # Kerberos hardening
        Write-Host "Hardening Kerberos Configuration..."
-ForegroundColor Yellow

        # Set Kerberos policies
        $KerberosPolicies = @{
            "MaxTicketAge" = 10  # Hours
            "MaxRenewAge" = 7    # Days
            "MaxServiceAge" = 600 # Minutes
            "MaxClockSkew" = 5   # Minutes
        }

        foreach ($Policy in $KerberosPolicies.GetEnumerator()) {
            Set-ADDefaultDomainPasswordPolicy -$($Policy.Key)
$($Policy.Value)
        }

        # Disable weak encryption types
        Set-ADAccountControl -UseDESKeyOnly $false

        # Configure SPN validation
        Set-ItemProperty -Path
"HKLM:\SYSTEM\CurrentControlSet\Services\Kdc" `
                         -Name "ValidateKdcPacSignature" -Value 1
    }

    # Configure Protected Users group
    Write-Host "Configuring Protected Users..." -ForegroundColor Yellow
    $ProtectedUsers = Get-ADGroup -Filter {Name -eq "Protected Users"}
-ErrorAction SilentlyContinue
    if ($ProtectedUsers) {
        $HighValueAccounts = Get-ADUser -Filter {
            (memberof -recursiveMatch "CN=Domain Admins,CN=Users,DC=*")
-or
            (memberof -recursiveMatch "CN=Enterprise
Admins,CN=Users,DC=*")
        }

        foreach ($Account in $HighValueAccounts) {
            Add-ADGroupMember -Identity "Protected Users" -Members
$Account -ErrorAction SilentlyContinue
        }
    }
```

```
    Write-Host "Active Directory Security Hardening Complete!"
-ForegroundColor Green
}
```

 # Run hardening

Harden-ADSecurity -EnableAdvancedAuditing -HardenKerberos

Performance and Scalability Considerations

Event volume estimation for AD monitoring:

Environment Size	DCs	Daily Event Volume	Storage (30 days)	SIEM Requirements
Small (< 1000 users)	2	50-100 GB	3 TB	Splunk/ELK
Medium (1000-5000)	4	200-500 GB	15 TB	Splunk Enterprise
Large (5000-25000)	8+	1-2 TB	60 TB	Splunk/QRadar
Enterprise (25000+)	20+	5+ TB	150+ TB	Splunk/Sentinel

Related Articles and Resources

- Microsoft Advanced Threat Analytics Documentation
- MITRE ATT&CK - Active Directory Attacks
- SANS Active Directory Security
- Active Directory Security Blog - adsecurity.org
- Microsoft Defender for Identity
- BloodHound - AD Attack Path Management
- Ping Castle - AD Security Assessment
- Purple Knight - AD Security Assessment Tool
- CISA Alert - Active Directory Targeting
- NSA/CSS - Detecting and Preventing Kerberos Abuse
- Red Canary - Active Directory Attack Detection
- CrowdStrike - Falcon for Active Directory
- Varonis - Active Directory Security Guide
- Quest - Active Directory Security Best Practices
- ERNW - Active Directory Security Assessment