Service Principal Credential Addition Incident Response Playbook

Graph API Context:

- **Resource:** Service Principal Credentials
- **Graph Endpoint:** `https://graph.microsoft.com/v1.0/servicePrincipals/{id}/addPassword`
- **AuditLogs Category:** ServicePrincipal credential management
- **Log Sources:** AuditLogs, MicrosoftGraphActivityLogs, AADServicePrincipalSignInLogs, AADServicePrincipalRiskEvents
- **Detection Scope:** Credential additions to existing service principals + API authentication patterns

Detection Summary: This playbook detects credentials added directly to Service Principal objects via Graph API - a stealthy persistence technique where credentials are hidden from Azure Portal. The detection focuses on secondary credential additions (excludes first credentials for new service principals) and correlates with sign-in activity to identify dormant service principal activation and anomalous access patterns.

Attack Technique: MITRE ATT&CK T1098.001 - Account Manipulation: Additional Cloud Credentials

Scope: This playbook addresses SERVICE PRINCIPAL credentials only. Application object credential additions require a separate playbook (different OperationName: "Update application – Certificates and secrets management").

DETECTION RULE

Deploy this query as a scheduled alert rule:

```
AuditLogs
| where OperationName == "Add service principal credentials"
| where TimeGenerated > ago(1h)  // Run hourly
// ✅ CRITICAL: Extract InitiatedBy fields BEFORE mv-expand
| extend
    Initiator = coalesce(
        tostring(InitiatedBy.user.userPrincipalName),
        tostring(InitiatedBy.app.displayName)
    ),
    ActorAppId = tostring(InitiatedBy.app.appId),
    SourceIP = tostring(InitiatedBy.user.ipAddress)
// Now safe to expand TargetResources
| mv-expand TargetResource = TargetResources
| extend
    SPDisplayName = tostring(TargetResource.displayName),
    SPObjectId = tostring(TargetResource.id)
// Now safe to expand modifiedProperties
| mv-expand ModifiedProperty = TargetResource.modifiedProperties
```

```
| extend
    PropertyName = tostring(ModifiedProperty.displayName),
    OldValue = tostring(ModifiedProperty.oldValue),
    NewValue = tostring(ModifiedProperty.newValue)
| where PropertyName == "KeyDescription"
| extend IsFirstCredential = (OldValue == "[]")
| where IsFirstCredential == false  // Exclude new SPs (first credential)
// ⚠ WHITELIST: Add known credential rotation accounts
| where Initiator !in (
    "automation@company.com",        // Credential rotation account
    "cicd-pipeline-sp",              // CI/CD service principal
    "azure-devops@company.com"       // DevOps admin
)
| project
    TimeGenerated,
    SPDisplayName,
    SPObjectId,
    Initiator,
    ActorAppId,
    SourceIP,
    Result,
    OldValue,
    NewValue,
    CorrelationId
| sort by TimeGenerated desc
```

Alert Configuration:

- **Rule Name:** Service Principal Secondary Credential Addition
- **Severity:** Medium (High if targeting privileged SPs)
- **Frequency:** Every 1 hour
- **Lookback Period:** 1 hour
- **Threshold:** 1 event
- **MITRE ATT&CK:** T1098.001 - Account Manipulation: Additional Cloud Credentials
- **Tactics:** Persistence, Privilege Escalation

Whitelisting: Update the `Initiator !in (...)` list with known legitimate accounts that perform credential rotation. Review quarterly.

ALERT TRIAGE

Initial Assessment Questions

When an alert fires, answer these questions first:

1. **Who added the credential?**
    - Check `Initiator` field - is it a user account or application?
    - Check `ActorAppId` if initiated by application
2. **What service principal was targeted?**

- Check `SPDisplayName` - is this a high-value or privileged application?
- Does this SP have admin roles or sensitive permissions?

3. **When did this occur?**
   - Business hours or after-hours?
   - Weekend or holiday?
4. **What was the result?**
   - `Result` = "success" or "failure"?
   - Failures may indicate reconnaissance
5. **What was the source location?**
   - Check `SourceIP` - expected corporate IP range?
   - VPN? Cloud provider? Foreign country?

Quick True Positive / False Positive Checks

Likely FALSE POSITIVE if:

- Initiator is in your whitelist (but verify it's not compromised)
- ActorAppId matches known DevOps/automation tools
- Activity during scheduled maintenance window
- Source IP from known corporate/datacenter range
- Same initiator has performed this operation before recently

Likely TRUE POSITIVE if:

- Dormant service principal suddenly activated after credential add
- High-risk country in sign-in activity immediately after
- Initiator account was recently compromised
- Service principal has privileged roles
- Activity outside business hours with no change ticket
- Multiple credential adds in short timeframe

Escalation Criteria

IMMEDIATE ESCALATION (Critical):

- Service principal has Global Administrator or other admin roles
- Identity Protection shows HIGH risk on SP sign-ins
- Dormant SP (never used before) now active
- Sign-ins from sanctioned countries (Russia, China, North Korea, Iran, Syria)
- Credential added + immediate access to sensitive resources (Exchange, SharePoint admin APIs)

STANDARD INVESTIGATION (High):

- Identity Protection shows MEDIUM risk
- New geographic locations in sign-in activity
- Multiple anomalies (new IPs + new resources + unusual timing)

- Cannot verify legitimacy with app owner

MONITORING ONLY (Medium):

- Identity Protection shows LOW/NO risk
- Normal baseline activity continues
- Verified legitimate with app owner
- No anomalous sign-ins detected

INVESTIGATION PROCEDURE

Step 1: Identify the Target Service Principal

Objective: Get basic information about the service principal and assess its privilege level.

```
let SPObjectId = "PASTE_SP_OBJECT_ID_FROM_ALERT";  // From alert's
SPObjectId field

AuditLogs
| where OperationName has_any ("Add member to role", "Add app role
assignment to service principal")
| where TimeGenerated > ago(90d)
| where TargetResources has SPObjectId
| extend
    Initiator = coalesce(
        tostring(InitiatedBy.user.userPrincipalName),
        tostring(InitiatedBy.app.displayName)
    )
| mv-expand TargetResource = TargetResources
| extend
    TargetId = tostring(TargetResource.id),
    TargetType = tostring(TargetResource.type),
    TargetDisplayName = tostring(TargetResource.displayName)
| where TargetId == SPObjectId
| mv-expand ModifiedProperty = TargetResource.modifiedProperties
| extend
    PropertyName = tostring(ModifiedProperty.displayName),
    NewValue = tostring(ModifiedProperty.newValue)
| where PropertyName in ("Role.DisplayName", "AppRole.Value")
| project
    TimeGenerated,
    OperationName,
    PropertyName,
    NewValue,
    Initiator,
    TargetDisplayName
| sort by TimeGenerated desc
```

What to look for:

- **Admin roles:** GlobalAdministrator, ApplicationAdministrator, CloudApplicationAdministrator, PrivilegedRoleAdministrator
- **High-value app permissions:** Mail.ReadWrite, Files.ReadWrite.All, Directory.ReadWrite.All
- **Recent permission grants:** Were permissions added shortly before credential?

Red flags:

- Privileged roles or permissions assigned
- Permissions granted within 24 hours of credential addition
- Permissions that allow data exfiltration or persistence

Step 2: Analyze Credential Addition Activity

Objective: Understand the context of who added the credential and look for suspicious patterns.

```
let SPObjectId = "PASTE_SP_OBJECT_ID_FROM_ALERT";
let LookbackDays = 30;

AuditLogs
| where OperationName in ("Add service principal credentials", "Add
application credentials", "Update service principal")
| where TimeGenerated > ago(LookbackDays)
| extend
    Initiator = coalesce(
        tostring(InitiatedBy.user.userPrincipalName),
        tostring(InitiatedBy.app.displayName)
    ),
    ActorAppId = tostring(InitiatedBy.app.appId),
    SourceIP = tostring(InitiatedBy.user.ipAddress)
| mv-expand TargetResource = TargetResources
| where tostring(TargetResource.id) == SPObjectId
| project
    TimeGenerated,
    OperationName,
    Initiator,
    ActorAppId,
    SourceIP,
    Result,
    CorrelationId
| sort by TimeGenerated desc
```

What to look for:

- **Multiple credential adds:** Pattern of adding many credentials quickly
- **Failed attempts:** Multiple failures before success (reconnaissance)
- **Same initiator:** Has this user/app added credentials to other SPs?
- **Source IP consistency:** Same IP or different IPs?

Red flags:

- Multiple credential adds to same SP in short timeframe
- Same initiator added credentials to multiple different SPs
- Failed attempts followed by success (indicates trial and error)
- VPN/proxy IP addresses
- IP addresses from cloud providers (AWS, Azure, GCP)

Step 3: Check Initiator's Recent Activity

Objective: Determine if the account that added the credential is compromised.

```
let InitiatorUPN = "PASTE_INITIATOR_FROM_ALERT";
let LookbackHours = 24;

// If initiator is a USER account
AuditLogs
| where TimeGenerated > ago(LookbackHours)
| extend
    ActorUPN = tostring(InitiatedBy.user.userPrincipalName),
    ActorAppId = tostring(InitiatedBy.app.appId)
| where ActorUPN == InitiatorUPN
| extend
    ResourceTarget = tostring(TargetResources[0].displayName),
    ResourceType = tostring(TargetResources[0].type)
| summarize
    Operations = make_set(OperationName),
    TargetCount = dcount(ResourceTarget),
    FirstActivity = min(TimeGenerated),
    LastActivity = max(TimeGenerated),
    SourceIPs = make_set(tostring(InitiatedBy.user.ipAddress))
    by ActorUPN
| project
    ActorUPN,
    FirstActivity,
    LastActivity,
    OperationCount = array_length(Operations),
    TargetCount,
    Operations,
    SourceIPs
```

What to look for:

- **Unusual operations:** Password resets, MFA changes, role assignments, OAuth consents
- **High volume:** Unusually high number of operations
- **Multiple IPs:** Sign-ins from multiple locations simultaneously
- **Privileged operations:** Creating users, adding admin roles, modifying security settings

Red flags:

- MFA modifications or password changes shortly before credential add

- OAuth consent grants to suspicious applications
- Operations across many different resources (spray activity)
- Multiple simultaneous IP addresses (impossible travel)

Step 4: Timeline Analysis - Correlate Credential Add with First Use

Objective: Determine how quickly the new credential was used after being added.

```
let SPObjectId = "PASTE_SP_OBJECT_ID_FROM_ALERT";
let CredentialAddTime = datetime(2025-01-15T14:30:00Z);  // From alert
TimeGenerated
let WindowHours = 48;

// Get the AppId (Client ID) for sign-in correlation
let AppIdMapping = AuditLogs
| where OperationName == "Add service principal credentials"
| where TimeGenerated between ((CredentialAddTime - 1h) ..
(CredentialAddTime + 1h))
| mv-expand TargetResource = TargetResources
| where tostring(TargetResource.id) == SPObjectId
| mv-expand ModifiedProperty = TargetResource.modifiedProperties
| where tostring(ModifiedProperty.displayName) == "ServicePrincipalNames"
| extend AppId = tostring(parse_json(tostring(ModifiedProperty.newValue))
[0])
| distinct AppId;

// Correlate with sign-in activity
AADServicePrincipalSignInLogs
| where AppId in (AppIdMapping)
| where TimeGenerated between ((CredentialAddTime - 2h) ..
(CredentialAddTime + WindowHours))
| extend
    TimeSinceCredAdd = TimeGenerated - CredentialAddTime,
    Country = tostring(LocationDetails.countryOrRegion),
    State = tostring(LocationDetails.state),
    City = tostring(LocationDetails.city)
| project
    TimeGenerated,
    TimeSinceCredAdd,
    ServicePrincipalName,
    IPAddress,
    Country,
    City,
    ResourceDisplayName,
    ResultType
| sort by TimeGenerated asc
```

What to look for:

- **First sign-in timing:** How soon after credential add did authentication occur?

- **Geographic location:** Where did the first sign-in originate from?
- **Resources accessed:** What APIs/resources were accessed immediately after?

Red flags:

- First sign-in within minutes of credential addition (automated attack)
- Sign-in from different country than credential addition
- Immediate access to sensitive resources (Exchange, SharePoint, OneDrive)
- Failed sign-ins before successful (testing credentials)

Step 5: Baseline Comparison - New vs. Historical Activity

Objective: Compare recent activity to historical baseline to identify anomalies.

```
let AppId = "PASTE_APP_ID_FROM_STEP_4";  // Client ID (AppId) from
timeline query
let BaselineDays = 30;
let RecentDays = 2;

AADServicePrincipalSignInLogs
| where AppId == AppId
| where TimeGenerated > ago(BaselineDays)
| extend
    Country = tostring(LocationDetails.countryOrRegion),
    IsRecent = TimeGenerated > ago(RecentDays)
| summarize
    BaselineSignIns = countif(IsRecent == false),
    RecentSignIns = countif(IsRecent == true),
    BaselineCountries = make_set_if(Country, IsRecent == false),
    RecentCountries = make_set_if(Country, IsRecent == true),
    BaselineIPs = make_set_if(IPAddress, IsRecent == false),
    RecentIPs = make_set_if(IPAddress, IsRecent == true),
    BaselineResources = make_set_if(ResourceDisplayName, IsRecent ==
false),
    RecentResources = make_set_if(ResourceDisplayName, IsRecent == true),
    FirstRecentActivity = minif(TimeGenerated, IsRecent == true),
    LastRecentActivity = maxif(TimeGenerated, IsRecent == true)
| extend
    Countries_Diff = set_difference(RecentCountries, BaselineCountries),
    IPs_Diff = set_difference(RecentIPs, BaselineIPs),
    Resources_Diff = set_difference(RecentResources, BaselineResources),
    IsDormantActivation = (BaselineSignIns == 0 and RecentSignIns > 0)
| project
    IsDormantActivation,
    BaselineSignIns,
    RecentSignIns,
    FirstRecentActivity,
    LastRecentActivity,
    BaselineCountries,
    RecentCountries,
    Countries_Diff,
```

```
    BaselineIPs,
    RecentIPs,
    IPs_Diff,
    BaselineResources,
    RecentResources,
    Resources_Diff
```

What to look for:

- **Dormant activation:** `IsDormantActivation = true` (baseline = 0, recent > 0)
- **Countries_Diff:** New countries that never appeared in baseline
- **IPs_Diff:** New IP addresses never seen before
- **Resources_Diff:** New APIs/resources being accessed

Critical findings:

- **Dormant SP activated** = STRONG indicator of compromise
- **New high-risk countries** (Russia, China, North Korea, Iran, Syria) = CONTAIN immediately
- **New sensitive resources** (Mail., *Files.*, Directory.*) = High priority investigation
- **All Diff columns empty** = Likely legitimate activity

SECONDARY LOG SOURCE ANALYSIS

MicrosoftGraphActivityLogs - API Call Patterns

Objective: Analyze specific Graph API calls made by the service principal after credential addition.

```
let AppId = "PASTE_APP_ID";  // Client ID from previous queries
let CredentialAddTime = datetime(2025-01-15T14:30:00Z);

MicrosoftGraphActivityLogs
| where AppId == AppId
| where TimeGenerated > CredentialAddTime
| where TimeGenerated < (CredentialAddTime + 7d)
| summarize
    CallCount = count(),
    UniqueOperations = make_set(OperationName),
    FirstCall = min(TimeGenerated),
    LastCall = max(TimeGenerated),
    TargetResources = make_set(RequestUri)
    by AppId, IPAddress
| extend TimeSinceCredAdd = FirstCall - CredentialAddTime
| project
    FirstCall,
    TimeSinceCredAdd,
    IPAddress,
    CallCount,
    UniqueOperations,
    TargetResources
| sort by FirstCall asc
```

What to look for:

- **High volume API calls:** Hundreds or thousands of calls in short period
- **Data exfiltration patterns:** Mail export, file downloads, user enumeration
- **Suspicious operations:** ListUsers, ListGroups, ReadMail, DownloadFiles
- **Timing:** API calls within minutes of credential addition

Red flags:

- Mass enumeration (ListUsers, ListGroups, ListApplications)
- Mail access (messages, mailFolders, attachments)
- File downloads from SharePoint/OneDrive
- Directory modifications (user creation, role assignments)

AADServicePrincipalSignInLogs - Authentication Patterns

Objective: Examine authentication success/failure patterns and geographic distribution.

```
let AppId = "PASTE_APP_ID";
let CredentialAddTime = datetime(2025-01-15T14:30:00Z);

AADServicePrincipalSignInLogs
| where AppId == AppId
| where TimeGenerated > (CredentialAddTime - 2h)
| where TimeGenerated < (CredentialAddTime + 7d)
| extend
    Country = tostring(LocationDetails.countryOrRegion),
    City = tostring(LocationDetails.city)
| summarize
    TotalSignIns = count(),
    SuccessCount = countif(ResultType == "0"),
    FailureCount = countif(ResultType != "0"),
    Countries = make_set(Country),
    Cities = make_set(City),
    IPAddresses = make_set(IPAddress),
    ResourcesAccessed = make_set(ResourceDisplayName),
    FirstSignIn = min(TimeGenerated),
    LastSignIn = max(TimeGenerated)
    by AppId
| extend
    FailureRate = round(100.0 * FailureCount / TotalSignIns, 2),
    TimeSinceCredAdd = FirstSignIn - CredentialAddTime
| project
    FirstSignIn,
    TimeSinceCredAdd,
    TotalSignIns,
    SuccessCount,
    FailureCount,
    FailureRate,
```

```
    Countries,
    Cities,
    IPAddresses,
    ResourcesAccessed
```

What to look for:

- **Failure patterns:** High failure rate may indicate credential testing
- **Geographic spread:** Sign-ins from multiple countries
- **Resource diversity:** Accessing many different APIs
- **Timing patterns:** Burst of activity vs. steady rate

Red flags:

- Multiple failed attempts before success
- Sign-ins from residential ISPs (non-corporate)
- Access to resources the SP never accessed before
- Geographic anomalies (impossible travel)

Identity Protection Risk Events (If Available)

Requirements: Microsoft Entra ID P2 + Workload Identities Premium

```
let AppId = "PASTE_APP_ID";

AADServicePrincipalRiskEvents
| where AppId == AppId
| where TimeGenerated > ago(14d)
| where RiskState !in ("dismissed", "remediated", "confirmedSafe")
| where RiskLevel in ("high", "medium")
| project
    TimeGenerated,
    RiskEventType,
    RiskLevel,
    RiskState,
    RiskDetail,
    ServicePrincipalDisplayName,
    Location = AdditionalInfo
| sort by TimeGenerated desc
```

Common risk detections:

- Anomalous service principal activity
- Suspicious API calls
- Atypical travel (geographic anomaly)
- Anonymous IP address usage
- Unfamiliar sign-in properties
- Mass access to sensitive resources

Action based on risk:

- **High Risk** → CONTAIN immediately (proceed to Containment section)
- **Medium Risk** → Complete full investigation before containment decision
- **Low/No Risk** → Focus on baseline comparison results

AZURE PORTAL INVESTIGATION

Step 1: Examine Service Principal in Portal

1. Navigate to **Entra ID** → **Enterprise Applications**
    - Direct URL: `https://portal.azure.com/#view/Microsoft_AAD_IAM/StartboardApplicationsMenuBlade/~/AppAppsPreview`
2. Search for the service principal by name or Application (Client) ID
3. Check **Properties** tab:
    - **Enabled for users to sign-in:** Should be "Yes" if active
    - **Assignment required:** Check if assignment is enforced
    - **Visible to users:** Check visibility settings
    - **Created date:** When was this SP created?
    - **Home tenant ID:** Verify it's your tenant
4. Check **Permissions** tab:
    - Click **Admin consent granted** to see all permissions
    - **Red flags:** Mail.ReadWrite, Files.ReadWrite.All, Directory.ReadWrite.All, RoleManagement.ReadWrite.Directory
    - Note: Date granted vs. credential addition date
5. Check **Activity** → **Sign-in logs**:
    - Look for recent sign-ins
    - Check IP addresses and locations
    - Review resources accessed

What you CANNOT see in Portal:

- Service Principal credentials (secrets/certificates) - these are hidden by design
- Which specific credential was added
- Credential expiration dates for SP credentials

Note: To view SP credentials, you must use CLI: `az ad sp credential list --id {sp-id}`

Step 2: Review Audit Logs in Portal

1. Navigate to **Entra ID** → **Audit logs**

- Direct URL: `https://portal.azure.com/#view/Microsoft_AAD_IAM/ActiveDirectoryMenuBlade/~/Audit`

2. Filter by:
   - **Activity:** "Add service principal credentials"
   - **Target:** Service principal name
   - **Date range:** Last 30 days
3. Click on the credential addition event:
   - Check **Initiated by** → who added it?
   - Check **Modified properties** → KeyDescription changes
   - Check **Target** → verify correct SP
   - Check **Additional details** → any unusual values?
4. Look for related events:
   - Filter **Activity:** "Add app role assignment to service principal"
   - Check if permissions were granted around same time
   - Filter **Activity:** "Update service principal"
   - Check for other modifications

Red flags:

- Credential added outside business hours
- Same actor modified multiple SPs
- Permissions granted within 24 hours of credential add
- Failed attempts before successful add

Step 3: Check Application Owner

1. In **Enterprise Applications**, select the service principal
2. Go to **Owners** tab:
   - **No owners listed:** Red flag - orphaned application
   - **Multiple owners:** Verify all are legitimate
   - **External users:** High risk if owners are guest accounts
3. Verify owner legitimacy:
   - Click on owner name
   - Check user's properties and last sign-in
   - Verify department and role
   - Contact owner to verify recent credential additions

If owner unresponsive or suspicious:

- Escalate to security team

- Check owner's recent activity in audit logs
- Consider owner account may be compromised

POWERSHELL INVESTIGATION

Run this consolidated script to gather all relevant information:

```
#
==========================================================================
==
# Service Principal Credential Addition Investigation Script
#
==========================================================================
==

Connect-MgGraph -Scopes "Application.Read.All", "AuditLog.Read.All",
"Directory.Read.All" -NoWelcome

#
==========================================================================
==
# CONFIGURATION - Set these variables
#
==========================================================================
==

$AppId = "YOUR_CLIENT_ID_HERE"  # Application (Client) ID from alert

#
==========================================================================
==
# 1. Get Service Principal Details
#
==========================================================================
==

Write-Host "`n[*] Getting Service Principal Details..." -ForegroundColor
Cyan

$SP = Get-MgServicePrincipal -Filter "appId eq '$AppId'"

if (-not $SP) {
    Write-Host "[!] Service Principal not found with AppId: $AppId" -
ForegroundColor Red
    exit
}

Write-Host "[+] Service Principal Found:" -ForegroundColor Green
Write-Host "    Display Name: $($SP.DisplayName)" -ForegroundColor White
Write-Host "    Object ID: $($SP.Id)" -ForegroundColor White
Write-Host "    App ID (Client ID): $($SP.AppId)" -ForegroundColor White
```

```
Write-Host "    Created: $($SP.CreatedDateTime)" -ForegroundColor White
Write-Host "    Enabled: $($SP.AccountEnabled)" -ForegroundColor White
Write-Host "    Publisher: $($SP.PublisherName)" -ForegroundColor White

#
==========================================================================
==
# 2. Check Service Principal Credentials
#
==========================================================================
==

Write-Host "`n[*] Checking Service Principal Credentials..." -
ForegroundColor Cyan

$SPCreds = $SP.PasswordCredentials
if ($SPCreds.Count -gt 0) {
    Write-Host "[!] Found $($SPCreds.Count) Password Credential(s):" -
ForegroundColor Yellow
    foreach ($Cred in $SPCreds) {
        Write-Host "    KeyId: $($Cred.KeyId)" -ForegroundColor White
        Write-Host "    DisplayName: $($Cred.DisplayName)" -
ForegroundColor White
        Write-Host "    Created: $($Cred.StartDateTime)" -ForegroundColor
White
        Write-Host "    Expires: $($Cred.EndDateTime)" -ForegroundColor
White
        Write-Host "    ---" -ForegroundColor Gray
    }
} else {
    Write-Host "[+] No password credentials found" -ForegroundColor Green
}

$SPCerts = $SP.KeyCredentials
if ($SPCerts.Count -gt 0) {
    Write-Host "[!] Found $($SPCerts.Count) Certificate Credential(s):" -
ForegroundColor Yellow
    foreach ($Cert in $SPCerts) {
        Write-Host "    KeyId: $($Cert.KeyId)" -ForegroundColor White
        Write-Host "    DisplayName: $($Cert.DisplayName)" -
ForegroundColor White
        Write-Host "    Created: $($Cert.StartDateTime)" -ForegroundColor
White
        Write-Host "    Expires: $($Cert.EndDateTime)" -ForegroundColor
White
        Write-Host "    ---" -ForegroundColor Gray
    }
} else {
    Write-Host "[+] No certificate credentials found" -ForegroundColor
Green
}

#
==========================================================================
```

```
==
# 3. Check Assigned Permissions (App Roles)
#
================================================================================
==

Write-Host "`n[*] Checking App Role Assignments..." -ForegroundColor Cyan

$AppRoles = Get-MgServicePrincipalAppRoleAssignment -ServicePrincipalId
$SP.Id

if ($AppRoles.Count -gt 0) {
    Write-Host "[!] Found $($AppRoles.Count) App Role Assignment(s):" -
ForegroundColor Yellow
    foreach ($Role in $AppRoles) {
        $ResourceSP = Get-MgServicePrincipal -ServicePrincipalId
$Role.ResourceId -ErrorAction SilentlyContinue
        Write-Host "    Resource: $($ResourceSP.DisplayName)" -
ForegroundColor White
        Write-Host "    Role ID: $($Role.AppRoleId)" -ForegroundColor
White
        Write-Host "    Created: $($Role.CreatedDateTime)" -
ForegroundColor White
        Write-Host "    ---" -ForegroundColor Gray
    }
} else {
    Write-Host "[+] No app role assignments found" -ForegroundColor Green
}

#
================================================================================
==
# 4. Check Directory Roles (Admin Roles)
#
================================================================================
==

Write-Host "`n[*] Checking Directory Role Memberships..." -ForegroundColor
Cyan

$SPMemberOf = Get-MgServicePrincipalMemberOf -ServicePrincipalId $SP.Id

$DirectoryRoles = $SPMemberOf | Where-Object { $_.'@odata.type' -eq
'#microsoft.graph.directoryRole' }

if ($DirectoryRoles.Count -gt 0) {
    Write-Host "[!] Found $($DirectoryRoles.Count) Directory Role(s):" -
ForegroundColor Red
    foreach ($Role in $DirectoryRoles) {
        $RoleDetails = Get-MgDirectoryRole -DirectoryRoleId $Role.Id
        Write-Host "    Role: $($RoleDetails.DisplayName)" -
ForegroundColor Red
        Write-Host "    Role ID: $($Role.Id)" -ForegroundColor White
        Write-Host "    Description: $($RoleDetails.Description)" -
```

```
ForegroundColor Gray
        Write-Host "    ---" -ForegroundColor Gray
    }
} else {
    Write-Host "[+] No directory roles assigned (good)" -ForegroundColor
Green
}

#
==========================================================================
==
# 5. Check Application Owners
#
==========================================================================
==

Write-Host "`n[*] Checking Application Owners..." -ForegroundColor Cyan

# Get the Application object (different from Service Principal)
$App = Get-MgApplication -Filter "appId eq '$AppId'"

if ($App) {
    $Owners = Get-MgApplicationOwner -ApplicationId $App.Id

    if ($Owners.Count -gt 0) {
        Write-Host "[+] Found $($Owners.Count) Owner(s):" -ForegroundColor
Green
        foreach ($Owner in $Owners) {
            Write-Host "    Owner ID: $($Owner.Id)" -ForegroundColor White
            # Try to get owner details
            $User = Get-MgUser -UserId $Owner.Id -ErrorAction
SilentlyContinue
            if ($User) {
                Write-Host "    Name: $($User.DisplayName)" -
ForegroundColor White
                Write-Host "    UPN: $($User.UserPrincipalName)" -
ForegroundColor White
            }
            Write-Host "    ---" -ForegroundColor Gray
        }
    } else {
        Write-Host "[!] No owners found - ORPHANED APPLICATION" -
ForegroundColor Red
    }
} else {
    Write-Host "[!] Could not find Application object" -ForegroundColor
Yellow
}

#
==========================================================================
==
# 6. Recent Sign-In Activity Summary
#
```

```
==========================================================================
==

Write-Host "`n[*] Checking Recent Sign-In Activity..." -ForegroundColor
Cyan
Write-Host "    Note: Sign-in logs must be queried separately via Azure
Portal or KQL" -ForegroundColor Gray
Write-Host "    Use AADServicePrincipalSignInLogs table in Log Analytics"
-ForegroundColor Gray

#
==========================================================================
==
# Investigation Complete
#
==========================================================================
==

Write-Host "`n[+] Investigation Complete" -ForegroundColor Green
Write-Host "[*] Next Steps:" -ForegroundColor Cyan
Write-Host "    1. Review credential timestamps against alert time" -
ForegroundColor White
Write-Host "    2. Check sign-in logs in Azure Portal" -ForegroundColor
White
Write-Host "    3. Verify owners and permissions are legitimate" -
ForegroundColor White
Write-Host "    4. Run baseline comparison query in Log Analytics" -
ForegroundColor White
```

CONTAINMENT ACTIONS

Prerequisites

Before containment, ensure you have:

1. **Evidence preserved** - Screenshot or export all credential info, permissions, and recent activity
2. **Impact assessment** - Understand which applications/services will be affected
3. **Approval** - Get approval from incident commander for production impact

Step 1: Preserve Evidence - Export Current State

```
Connect-MgGraph -Scopes "Application.Read.All", "AuditLog.Read.All" -
NoWelcome

$AppId = "YOUR_CLIENT_ID"
$SP = Get-MgServicePrincipal -Filter "appId eq '$AppId'"

# Export service principal details
$Evidence = @{
    DisplayName = $SP.DisplayName
```

```
    AppId = $SP.AppId
    ObjectId = $SP.Id
    AccountEnabled = $SP.AccountEnabled
    CreatedDateTime = $SP.CreatedDateTime
    PasswordCredentials = $SP.PasswordCredentials | Select-Object KeyId,
DisplayName, StartDateTime, EndDateTime
    KeyCredentials = $SP.KeyCredentials | Select-Object KeyId,
DisplayName, StartDateTime, EndDateTime
    Timestamp = Get-Date
}

$Evidence | ConvertTo-Json -Depth 10 | Out-File
"SP_Evidence_$($SP.AppId)_$(Get-Date -Format 'yyyyMMdd_HHmmss').json"
Write-Host "[+] Evidence saved to SP_Evidence_$($SP.AppId)_$(Get-Date -
Format 'yyyyMMdd_HHmmss').json" -ForegroundColor Green
```

Step 2: Revoke All Active Sessions

```
Connect-MgGraph -Scopes "Application.ReadWrite.All" -NoWelcome

$AppId = "YOUR_CLIENT_ID"
$SP = Get-MgServicePrincipal -Filter "appId eq '$AppId'"

Write-Host "[!] Revoking all sign-in sessions for $($SP.DisplayName)..." -
ForegroundColor Yellow

# Revoke sign-in sessions (invalidates issued tokens)
Revoke-MgServicePrincipalSignInSession -ServicePrincipalId $SP.Id

Write-Host "[+] All sessions revoked - existing tokens invalidated" -
ForegroundColor Green
Write-Host "[!] Warning: Any applications using this SP will need to re-
authenticate" -ForegroundColor Yellow
```

Impact: This invalidates all currently issued access tokens. Applications will fail on next API call until they re-authenticate with valid credentials.

Step 3: Identify and Remove Malicious Credential

Important: Only remove the specific malicious credential if you can identify it. Do NOT remove all credentials without coordination with application owners.

```
Connect-MgGraph -Scopes "Application.ReadWrite.All" -NoWelcome

$AppId = "YOUR_CLIENT_ID"
$SP = Get-MgServicePrincipal -Filter "appId eq '$AppId'"
```

```
# List all current credentials
Write-Host "[*] Current Service Principal Credentials:" -ForegroundColor
Cyan
$SP.PasswordCredentials | Select-Object KeyId, DisplayName, StartDateTime,
EndDateTime | Format-Table

# To remove a specific credential by KeyId
$MaliciousKeyId = "PASTE_KEY_ID_HERE"  # KeyId of the malicious credential

Write-Host "[!] Removing credential with KeyId: $MaliciousKeyId" -
ForegroundColor Red

# Remove the specific credential
Remove-MgServicePrincipalPassword -ServicePrincipalId $SP.Id -KeyId
$MaliciousKeyId

Write-Host "[+] Malicious credential removed" -ForegroundColor Green

# Verify removal
$SPUpdated = Get-MgServicePrincipal -ServicePrincipalId $SP.Id
Write-Host "[*] Remaining credentials:
$($SPUpdated.PasswordCredentials.Count)" -ForegroundColor Cyan
```

How to identify malicious credential:

1. Compare credential `StartDateTime` with alert `TimeGenerated`
2. The credential added at the alert time is the suspicious one
3. Remove only that specific `KeyId`

Impact: Only the removed credential stops working. Other credentials remain functional.

Step 4: Disable Service Principal (High-Risk Cases Only)

Use this only if:

- SP has privileged roles (Global Admin, Application Admin, etc.)
- Dormant SP suddenly activated
- Unable to identify which credential is malicious
- Active data exfiltration detected

```
Connect-MgGraph -Scopes "Application.ReadWrite.All" -NoWelcome

$AppId = "YOUR_CLIENT_ID"
$SP = Get-MgServicePrincipal -Filter "appId eq '$AppId'"

Write-Host "[!] DISABLING Service Principal: $($SP.DisplayName)" -
ForegroundColor Red
Write-Host "[!] This will break ALL applications using this SP" -
ForegroundColor Yellow
```

```
$confirm = Read-Host "Type 'DISABLE' to confirm"

if ($confirm -eq "DISABLE") {
    Update-MgServicePrincipal -ServicePrincipalId $SP.Id -
AccountEnabled:$false
    Write-Host "[+] Service Principal disabled successfully" -
ForegroundColor Green
    Write-Host "[!] Coordinate with app owners for re-enablement after
investigation" -ForegroundColor Yellow
} else {
    Write-Host "[*] Disable cancelled" -ForegroundColor Cyan
}
```

Impact:

- ⚠ **CRITICAL:** ALL authentication attempts will fail immediately
- All applications using this SP will stop working
- Use only when absolutely necessary
- Coordinate with application owners before re-enabling

Step 5: Remove Sensitive Permissions (If Applicable)

If the SP has privileged roles or dangerous permissions:

```
Connect-MgGraph -Scopes "RoleManagement.ReadWrite.Directory",
"AppRoleAssignment.ReadWrite.All" -NoWelcome

$AppId = "YOUR_CLIENT_ID"
$SP = Get-MgServicePrincipal -Filter "appId eq '$AppId'"

# Remove Directory Roles (e.g., Global Administrator)
$RoleAssignments = Get-MgServicePrincipalMemberOf -ServicePrincipalId
$SP.Id |
    Where-Object { $_.'@odata.type' -eq '#microsoft.graph.directoryRole' }

foreach ($Role in $RoleAssignments) {
    $RoleDetails = Get-MgDirectoryRole -DirectoryRoleId $Role.Id
    Write-Host "[!] Removing directory role: $($RoleDetails.DisplayName)"
-ForegroundColor Yellow

    Remove-MgDirectoryRoleMemberByRef -DirectoryRoleId $Role.Id -
DirectoryObjectId $SP.Id
    Write-Host "[+] Removed role: $($RoleDetails.DisplayName)" -
ForegroundColor Green
}

# Remove dangerous app role assignments (Mail.ReadWrite,
Files.ReadWrite.All, etc.)
$AppRoles = Get-MgServicePrincipalAppRoleAssignment -ServicePrincipalId
$SP.Id
```

```
foreach ($AppRole in $AppRoles) {
    Write-Host "[*] App Role ID: $($AppRole.AppRoleId)" -ForegroundColor
Cyan
    Write-Host "    Resource: $($AppRole.ResourceDisplayName)" -
ForegroundColor White

    # Review each and decide if it should be removed
    # Remove-MgServicePrincipalAppRoleAssignment -ServicePrincipalId
$SP.Id -AppRoleAssignmentId $AppRole.Id
}
```

Step 6: Verify Containment

```
Connect-MgGraph -Scopes "Application.Read.All" -NoWelcome

$AppId = "YOUR_CLIENT_ID"
$SP = Get-MgServicePrincipal -Filter "appId eq '$AppId'"

Write-Host "`n[*] Post-Containment Verification:" -ForegroundColor Cyan
Write-Host "    Account Enabled: $($SP.AccountEnabled)" -ForegroundColor
$(if ($SP.AccountEnabled) { "Red" } else { "Green" })
Write-Host "    Password Credentials: $($SP.PasswordCredentials.Count)" -
ForegroundColor White
Write-Host "    Certificate Credentials: $($SP.KeyCredentials.Count)" -
ForegroundColor White

Write-Host "`n[+] Containment verification complete" -ForegroundColor
Green
Write-Host "[*] Monitor for attempted sign-ins over next 24 hours" -
ForegroundColor Cyan
```

REMEDIATION STEPS

Post-Containment Analysis

After containment, complete these remediation steps:

Step 1: Investigate the Initiator Account

If credential was added by a USER account:

```
let InitiatorUPN = "user@company.com";
let LookbackDays = 7;

AuditLogs
| where TimeGenerated > ago(LookbackDays)
```

```
| extend ActorUPN = tostring(InitiatedBy.user.userPrincipalName)
| where ActorUPN == InitiatorUPN
| summarize
    OperationTypes = make_set(OperationName),
    TargetsAffected = dcount(tostring(TargetResources[0].displayName)),
    FirstActivity = min(TimeGenerated),
    LastActivity = max(TimeGenerated)
    by ActorUPN, OperationName
| order by LastActivity desc
```

Check for signs of account compromise:

- MFA changes
- Password resets
- OAuth consent grants to unknown apps
- Multiple failed sign-ins before success
- Unusual geographic locations

If initiator account is compromised:

1. Reset user's password
2. Revoke all sessions
3. Require MFA re-registration
4. Check for other persistence mechanisms (OAuth apps, app credentials)

Step 2: Check for Additional Backdoors

Look for other credentials added by same initiator:

```
let InitiatorUPN = "user@company.com";
let LookbackDays = 30;

AuditLogs
| where TimeGenerated > ago(LookbackDays)
| where OperationName in (
    "Add service principal credentials",
    "Update application – Certificates and secrets management",
    "Add owner to application",
    "Add owner to service principal"
)
| extend
    ActorUPN = coalesce(
        tostring(InitiatedBy.user.userPrincipalName),
        tostring(InitiatedBy.app.displayName)
    )
| where ActorUPN == InitiatorUPN
| mv-expand TargetResource = TargetResources
| project
    TimeGenerated,
    OperationName,
```

```
    TargetDisplayName = tostring(TargetResource.displayName),
    TargetId = tostring(TargetResource.id),
    Result
| sort by TimeGenerated desc
```

What to look for:

- Multiple applications/SPs modified
- Owner additions (alternate persistence)
- Application credential additions (separate from SP credentials)

For each suspicious activity:

1. Run investigation procedure for that resource
2. Apply same containment measures
3. Document all affected resources

Step 3: Credential Rotation Decision

Scenario A - Single Credential Removed (Most Common):

If you successfully identified and removed only the malicious credential:

- **Action:** No additional credential rotation needed
- **Rationale:** Legitimate credentials remain functional, application continues working
- **Monitoring:** Watch for any authentication attempts with removed credential (should fail)
- **Duration:** Monitor for 24-48 hours

Scenario B - Full Credential Rotation (High-Risk Cases):

Rotate ALL credentials if:

- Cannot determine which credential was malicious
- SP has privileged roles
- SP accessed sensitive data
- Low confidence in containment

```
Connect-MgGraph -Scopes "Application.ReadWrite.All" -NoWelcome

$AppId = "YOUR_CLIENT_ID"
$SP = Get-MgServicePrincipal -Filter "appId eq '$AppId'"

Write-Host "[!] FULL CREDENTIAL ROTATION - This will break applications" -
ForegroundColor Red

# Remove ALL existing credentials
foreach ($Cred in $SP.PasswordCredentials) {
    Write-Host "[*] Removing credential: $($Cred.KeyId)" -ForegroundColor
Yellow
    Remove-MgServicePrincipalPassword -ServicePrincipalId $SP.Id -KeyId
```

```
$Cred.KeyId
}

Write-Host "[+] All credentials removed" -ForegroundColor Green

# Create new credential
$NewCred = Add-MgServicePrincipalPassword -ServicePrincipalId $SP.Id -
PasswordCredential @{
    DisplayName = "Rotated-$(Get-Date -Format 'yyyyMMdd')"
}

Write-Host "[+] New credential created:" -ForegroundColor Green
Write-Host "    Client ID: $AppId" -ForegroundColor Cyan
Write-Host "    Client Secret: $($NewCred.SecretText)" -ForegroundColor
Yellow
Write-Host "[!] Securely share this with application owner" -
ForegroundColor Red
```

Impact: ⚠ Applications will break until updated with new credentials. Coordinate with app owners.

Step 4: Re-enable Service Principal (If Disabled)

Only after completing Steps 1-3 and confirming no additional threats:

```
Connect-MgGraph -Scopes "Application.ReadWrite.All" -NoWelcome

$AppId = "YOUR_CLIENT_ID"
$SP = Get-MgServicePrincipal -Filter "appId eq '$AppId'"

if ($SP.AccountEnabled -eq $false) {
    Write-Host "[*] Service Principal currently disabled" -ForegroundColor
Yellow
    Write-Host "[*] Verify investigation complete:" -ForegroundColor
Yellow
    Write-Host "    ✓ Initiator identified and contained" -ForegroundColor
Gray
    Write-Host "    ✓ No additional backdoors found" -ForegroundColor Gray
    Write-Host "    ✓ Credentials rotated (if needed)" -ForegroundColor
Gray

    $proceed = Read-Host "Re-enable Service Principal? (y/n)"

    if ($proceed -eq 'y') {
        Update-MgServicePrincipal -ServicePrincipalId $SP.Id -
AccountEnabled:$true
        Write-Host "[+] SP re-enabled - monitor for 24 hours" -
ForegroundColor Green
    }
}
```

Step 5: Verify Data Access

If sensitive resources were accessed, determine impact:

```
let AppId = "PASTE_APP_ID";
let IncidentStart = datetime(2025-01-15T14:00:00Z);
let IncidentEnd = datetime(2025-01-15T18:00:00Z);

MicrosoftGraphActivityLogs
| where AppId == AppId
| where TimeGenerated between (IncidentStart .. IncidentEnd)
| where RequestUri has_any ("mail", "messages", "files", "drive", "users")
| summarize
    RequestCount = count(),
    Operations = make_set(OperationName),
    Resources = make_set(RequestUri)
    by IPAddress
| project IPAddress, RequestCount, Operations, Resources
```

Assess data exposure:

- **Mail access:** Notify affected users, consider email recall
- **File downloads:** Identify files accessed, assess sensitivity
- **User enumeration:** Assume attacker has user list, increase monitoring
- **Directory modifications:** Review and revert unauthorized changes

---

PREVENTION RECOMMENDATIONS

Detection Rules to Deploy

1. **This playbook's detection rule** (primary)
2. **Application credential additions** (separate attack vector):

   ```
   AuditLogs
   | where OperationName == "Update application – Certificates and
   secrets management"
   | where Result == "success"
   ```

3. **Dormant service principal activation:**

   ```
   AADServicePrincipalSignInLogs
   | summarize FirstSeen = min(TimeGenerated), RecentSeen =
   max(TimeGenerated) by AppId
   | where datetime_diff('day', RecentSeen, FirstSeen) > 90
   | where RecentSeen > ago(1d)
   ```

4. **Service principal with admin roles:**

```
AuditLogs
| where OperationName == "Add member to role"
| where TargetResources has "ServicePrincipal"
| mv-expand TargetResource = TargetResources
| where tostring(TargetResource.type) == "ServicePrincipal"
```

5. **Identity Protection risk events** (if available):

```
AADServicePrincipalRiskEvents
| where RiskLevel in ("high", "medium")
| where RiskState !in ("dismissed", "remediated")
```

Policies to Enable

1. **Conditional Access for Workload Identities:**
   - Require trusted locations for admin operations
   - Block legacy authentication protocols
   - Require compliant devices (where applicable)
2. **Identity Protection for Workload Identities:**
   - Enable automated risk detection
   - Configure risk-based policies
   - Alert on medium/high risk events
3. **Privileged Identity Management (PIM):**
   - Just-in-time admin role activation
   - Time-limited assignments
   - Approval workflows for sensitive roles
4. **Application Management:**
   - Enforce application ownership
   - Regular application access reviews
   - Disable unused applications
   - Limit service principal credential lifetimes

Monitoring to Configure

1. **Baseline monitoring:**

- Track normal service principal authentication patterns
- Document expected resources accessed
- Record typical geographic locations
- Maintain inventory of all service principals

2. **Anomaly detection:**
   - Alert on dormant SP activation
   - Alert on new geographic locations
   - Alert on sensitive resource access
   - Alert on high-volume API calls
3. **Regular reviews:**
   - Quarterly service principal audit
   - Review credential expiration dates
   - Verify application owners
   - Remove orphaned applications
4. **Integration:**
   - Forward Entra ID logs to SIEM
   - Correlate with endpoint/network logs
   - Alert on credential theft IOCs
   - Track credential usage in identity systems

QUICK REFERENCE

Triage Decision (60 seconds)

Option 1 - Identity Protection (if available):

1. Check `AADServicePrincipalSignInLogs` for `RiskLevel`
2. Check `AADServicePrincipalRiskEvents`
3. **High Risk** → CONTAIN immediately
4. **Medium Risk** → INVESTIGATE
5. **Low/None** → Use Option 2

Option 2 - Manual Baseline:

1. Run baseline comparison query with AppId
2. Check `_Diff` columns (Countries_Diff, IPs_Diff, Resources_Diff)
3. **CONTAIN if:**
   - Dormant SP (BaselineSignIns = 0)
   - High-risk countries (RU, CN, KP, IR, SY)
   - Cannot verify legitimacy
4. **INVESTIGATE if:**
   - New sensitive resources
   - Multiple anomalies

5. **CLOSE if:**
    - Normal activity (empty _Diff columns)
    - Verified with app owner

Containment Sequence (15 minutes)

```
1. Preserve evidence (2 min) - Export SP state to JSON
2. Revoke sessions (1 min) - Invalidate tokens
3. Identify malicious credential (2 min) - Match timestamps
4. Remove malicious credential (2 min) - Delete specific KeyId
5. Disable SP if needed (2 min) - High-risk cases only
6. Verify containment (1 min) - Check status
```

Investigation Queries (Copy-Paste Ready)

Get AppId from ObjectId:

```
let SPObjectId = "PASTE_HERE";
AuditLogs
| where OperationName == "Add service principal credentials"
| mv-expand TargetResource = TargetResources
| where tostring(TargetResource.id) == SPObjectId
| mv-expand ModifiedProperty = TargetResource.modifiedProperties
| where tostring(ModifiedProperty.displayName) == "ServicePrincipalNames"
| extend AppId = tostring(parse_json(tostring(ModifiedProperty.newValue))
[0])
| distinct AppId
```

Quick baseline check:

```
let AppId = "PASTE_HERE";
AADServicePrincipalSignInLogs
| where AppId == AppId
| summarize Count=count(),
Countries=make_set(tostring(LocationDetails.countryOrRegion))
```

PowerShell one-liner:

```
# Get SP by AppId
$SP = Get-MgServicePrincipal -Filter "appId eq 'PASTE_HERE'"

# List credentials
$SP.PasswordCredentials | Select KeyId, DisplayName, StartDateTime
```

Incident Documentation Template

```
INCIDENT: Service Principal Credential Addition
Date: [YYYY-MM-DD HH:MM UTC]
Analyst: [Your Name]
Severity: [Critical/High/Medium/Low]

SERVICE PRINCIPAL:
- Display Name: [Name]
- App ID (Client ID): [GUID]
- Object ID: [GUID]
- Baseline: [Active/Dormant/Never Used]

DETECTION:
- Alert Time: [Timestamp]
- Credential Added By: [User/App]
- Source IP: [IP Address]
- Location: [Country/City]

INVESTIGATION FINDINGS:
- Dormant Activation: [Yes/No]
- New Geographies: [List]
- New Resources: [List]
- Identity Protection Risk: [High/Medium/Low/None]
- Privileged Roles: [List if any]

DECISION: [CONTAIN / INVESTIGATE / CLOSE]
Justification: [1-2 sentences]

CONTAINMENT ACTIONS:
[ ] Evidence preserved
[ ] Sessions revoked
[ ] Malicious credential removed (KeyId: ___)
[ ] SP disabled (if applicable)
[ ] Permissions removed (if applicable)

REMEDIATION:
[ ] Initiator investigated/contained
[ ] Additional backdoors checked
[ ] Full credential rotation (if needed)
[ ] SP re-enabled (if applicable)
[ ] Data access assessed

OUTCOME:
[Was it malicious? What was the impact? Any data accessed?]

FOLLOW-UP:
[Any pending actions, lessons learned, or detection improvements]
```

DEPLOYMENT CHECKLIST

Pre-Deployment (15 minutes):

- [ ] Review detection query
- [ ] Add whitelisted accounts (automation, DevOps)
- [ ] Test query against last 7 days (baseline)
- [ ] Configure alert (hourly, threshold=1, severity=Medium)
- [ ] Assign to SOC analyst queue
- [ ] Brief SOC team on triage process

Day 1:

- [ ] Monitor first alerts
- [ ] Triage using Identity Protection or baseline method
- [ ] Adjust whitelist if excessive false positives
- [ ] Document initial patterns

Week 1:

- [ ] Calculate false positive rate
- [ ] Refine whitelist based on observed patterns
- [ ] Update documentation with environment-specific notes
- [ ] Train additional analysts

Month 1:

- [ ] Review incident metrics (true positive rate, MTTR)
- [ ] Adjust severity based on actual impact
- [ ] Update runbook with lessons learned
- [ ] Schedule quarterly whitelist review

SUPPORT & TROUBLESHOOTING

Q: How do I get the AppId (Client ID) if I only have the Object ID? A: Use the KQL query in Quick Reference section "Get AppId from ObjectId"

Q: Don't have Identity Protection? A: Use Option 2 (Manual Baseline Analysis) - works for all customers

Q: High volume of alerts? A: Add more entries to whitelist. Focus on dormant/never-used SPs first (highest risk)

Q: Can't identify which credential is malicious? A: Compare credential `StartDateTime` with alert `TimeGenerated`. The times should match closely.

Q: Service Principal credentials not visible in Portal? A: By design. Use PowerShell `Get-MgServicePrincipal` or Azure CLI `az ad sp credential list`

Q: Need to re-enable SP after containment? A: Follow Step 4 in Remediation section. Verify investigation complete first.

Q: Application broke after containment? A: If you removed wrong credential or disabled SP, coordinate with app owner to create new credential or re-enable

---

TECHNICAL NOTES

Why Service Principal vs Application credentials?

- Service Principal credentials added via Graph API (`/servicePrincipals/{id}/addPassword`)
- Application credentials added via Portal or API (`/applications/{id}/addPassword`)
- Different AuditLogs OperationName values
- Service Principal credentials more stealthy (not visible in Portal)
- Requires separate detection rules for each

Related Attack Vectors:

- Application object credential additions - `OperationName == "Update application – Certificates and secrets management"`
- Certificate-based authentication - `KeyCredentials` instead of `PasswordCredentials`
- Federated identity credentials - Separate persistence mechanism
- OAuth consent grants - Backdoor via delegated permissions

Identity Protection Details:

- Requires: Microsoft Entra ID P2 + Workload Identities Premium
- Machine learning-based risk detection
- Automated scoring (high/medium/low)
- Common detections: anomalous activity, atypical travel, anonymous IPs
- More info: https://learn.microsoft.com/entra/id-protection/concept-workload-identity-risk

Credential Types:

- **PasswordCredentials:** Client secrets, typically 1-2 years validity
- **KeyCredentials:** Certificates, can be 1-3 years validity
- Both can be used for client credential flow authentication

END OF PLAYBOOK

Version: 2.0
Last Updated: 2025-01-28
Maintenance: Quarterly whitelist review
Scope: Service Principal credentials only (separate playbook for Application credentials)