Windows Security Event Log  
  
// Failed Authentication (RDP, SMB)

SecurityEvent

| where EventID == 4625

| where TimeGenerated > ago(15m)

// Authentication Success (RDP, SMB)

SecurityEvent

| where EventID == 4624

| where TimeGenerated > ago(15m)

// Brute Force Attempt

SecurityEvent

| where EventID == 4625

| where TimeGenerated > ago(60m)

| summarize FailureCount = count() by SourceIP = IpAddress, EventID, Activity

| where FailureCount >= 10

// Brute Force Success Windows

let FailedLogons = SecurityEvent

| where EventID == 4625 and LogonType == 3

| where TimeGenerated > ago(60m)

| summarize FailureCount = count() by AttackerIP = IpAddress, EventID, Activity, LogonType, DestinationHostName = Computer

| where FailureCount >= 5;

let SuccessfulLogons = SecurityEvent

| where EventID == 4624 and LogonType == 3

| where TimeGenerated > ago(60m)

| summarize SuccessfulCount = count() by AttackerIP = IpAddress, LogonType, DestinationHostName = Computer, AuthenticationSuccessTime = TimeGenerated;

SuccessfulLogons

| join kind = leftouter FailedLogons on DestinationHostName, AttackerIP, LogonType

| project AuthenticationSuccessTime, AttackerIP, DestinationHostName, FailureCount, SuccessfulCount  
  
Linux Syslog  
  
// Failed logon (ip address extract)

let IpAddress\_REGEX\_PATTERN = @"\b\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}\b";

Syslog

| where Facility == "auth"

| where SyslogMessage startswith "Failed password for"

| project TimeGenerated, SourceIP = extract(IpAddress\_REGEX\_PATTERN, 0, SyslogMessage), DestinationHostName = HostName, DestinationIP = HostIP, Facility, SyslogMessage, ProcessName, SeverityLevel, Type

// Successful logon (ip address extract)

let IpAddress\_REGEX\_PATTERN = @"\b\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}\b";

Syslog

| where Facility == "auth"

| where SyslogMessage startswith "Accepted password for"

| project TimeGenerated, SourceIP = extract(IpAddress\_REGEX\_PATTERN, 0, SyslogMessage), DestinationHostName = HostName, DestinationIP = HostIP, Facility, SyslogMessage, ProcessName, SeverityLevel, Type

// Brute Force Attempt Linux Syslog

let IpAddress\_REGEX\_PATTERN = @"\b\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}\b";

Syslog

| where Facility == "auth" and SyslogMessage startswith "Failed password for"

| where TimeGenerated > ago(1h)

| project TimeGenerated, AttackerIP = extract(IpAddress\_REGEX\_PATTERN, 0, SyslogMessage), DestinationHostName = HostName, DestinationIP = HostIP, Facility, SyslogMessage, ProcessName, SeverityLevel, Type

| summarize FailureCount = count() by AttackerIP, DestinationHostName, DestinationIP

| where FailureCount >= 5

// Brute Force Success Linux

let FailedLogons = Syslog

| where Facility == "auth" and SyslogMessage startswith "Failed password for"

| where TimeGenerated > ago(1h)

| project TimeGenerated, SourceIP = extract(@"\b\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}\b", 0, SyslogMessage), DestinationHostName = HostName, DestinationIP = HostIP, Facility, SyslogMessage, ProcessName, SeverityLevel, Type

| summarize FailureCount = count() by AttackerIP = SourceIP, DestinationHostName

| where FailureCount >= 5;

let SuccessfulLogons = Syslog

| where Facility == "auth" and SyslogMessage startswith "Accepted password for"

| where TimeGenerated > ago(1h)

| project TimeGenerated, SourceIP = extract(@"\b\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}\b", 0, SyslogMessage), DestinationHostName = HostName, DestinationIP = HostIP, Facility, SyslogMessage, ProcessName, SeverityLevel, Type

| summarize SuccessfulCount = count() by SuccessTime = TimeGenerated, AttackerIP = SourceIP, DestinationHostName

| where SuccessfulCount >= 1

| project DestinationHostName, SuccessfulCount, AttackerIP, SuccessTime;

let BruteForceSuccesses = SuccessfulLogons

| join kind = leftouter FailedLogons on AttackerIP, DestinationHostName;

BruteForceSuccesses

// Queries the linux syslog for any user accounts created

// By @slendymayne (Discord)

Syslog

| where Facility == "authpriv" and SeverityLevel == "info"

| where SyslogMessage contains "new user" and SyslogMessage contains "shell=/bin/bash"

| project TimeGenerated, HostIP, HostName, ProcessID, SyslogMessage

// Queries for any users given sudo privileges

// By @slendymayne (Discord)

Syslog

| where Facility == "authpriv" and SeverityLevel == "info"

| where SyslogMessage contains "to group 'sudo'"

| project TimeGenerated, HostIP, Computer, ProcessID, SyslogMessage  
  
Azure Active Directory  
  
// View Mass AAD Auth Failures

SigninLogs

| where ResultDescription == "Invalid username or password or Invalid on-premise username or password."

| extend location = parse\_json(LocationDetails)

| extend City = location.city, State = location.state, Country = location.countryOrRegion, Latitude = location.geoCoordinates.latitude, Longitude = location.geoCoordinates.longitude

| project TimeGenerated, ResultDescription, UserPrincipalName, AppDisplayName, IPAddress, IPAddressFromResourceProvider, City, State, Country, Latitude, Longitude

// View Global Administrator Assignment

AuditLogs

| where OperationName == "Add member to role" and Result == "success"

| where TargetResources[0].modifiedProperties[1].newValue == '"Global Administrator"' or TargetResources[0].modifiedProperties[1].newValue == '"Company Administrator"'

| order by TimeGenerated desc

| project TimeGenerated, OperationName, AssignedRole = TargetResources[0].modifiedProperties[1].newValue, Status = Result, TargetResources

// View Password Activities

AuditLogs

| where OperationName contains "password"

| order by TimeGenerated

// Brute Force Success Azure Active Directory

let FailedLogons = SigninLogs

| where Status.failureReason == "Invalid username or password or Invalid on-premise username or password."

| where TimeGenerated > ago(1h)

| project TimeGenerated, Status = Status.failureReason, UserPrincipalName, UserId, UserDisplayName, AppDisplayName, AttackerIP = IPAddress, IPAddressFromResourceProvider, City = LocationDetails.city, State = LocationDetails.state, Country = LocationDetails.country, Latitude = LocationDetails.geoCoordinates.latitude, Longitude = LocationDetails.geoCoordinates.longitude

| summarize FailureCount = count() by AttackerIP, UserPrincipalName;

let SuccessfulLogons = SigninLogs

| where Status.errorCode == 0

| where TimeGenerated > ago(1h)

| project TimeGenerated, Status = Status.errorCode, UserPrincipalName, UserId, UserDisplayName, AppDisplayName, AttackerIP = IPAddress, IPAddressFromResourceProvider, City = LocationDetails.city, State = LocationDetails.state, Country = LocationDetails.country, Latitude = LocationDetails.geoCoordinates.latitude, Longitude = LocationDetails.geoCoordinates.longitude

| summarize SuccessCount = count() by AuthenticationSuccessTime = TimeGenerated, AttackerIP, UserPrincipalName, UserId, UserDisplayName;

let BruteForceSuccesses = SuccessfulLogons

| join kind = leftouter FailedLogons on AttackerIP, UserPrincipalName;

BruteForceSuccesses

| project AttackerIP, TargetAccount = UserPrincipalName, UserId, FailureCount, SuccessCount, AuthenticationSuccessTime

// Excessive password Resets

AuditLogs

| where OperationName startswith "Change" or OperationName startswith "Reset"

| order by TimeGenerated

| summarize count() by tostring(InitiatedBy)

| project Count = count\_, InitiatorId = parse\_json(InitiatedBy).user.id, InitiatorUpn = parse\_json(InitiatedBy).user.userPrincipalName, InitiatorIpAddress = parse\_json(InitiatedBy).user.ipAddress

| where Count >= 10  
  
  
Azure Storage Account  
  
// Authorization Error

StorageBlobLogs

| where MetricResponseType endswith "Error"

| where StatusText == "AuthorizationPermissionMismatch"

| order by TimeGenerated asc

// Reading a bunch of blobs

StorageBlobLogs

| where OperationName == "GetBlob"

//Deleting a bunch of blobs (in a short time period)

StorageBlobLogs | where OperationName == "DeleteBlob"

| where TimeGenerated > ago(24h)

//Putting a bunch of blobs (in a short time period)

StorageBlobLogs | where OperationName == "PutBlob"

| where TimeGenerated > ago(24h)

//Copying a bunch of blobs (in a short time period)

StorageBlobLogs | where OperationName == "CopyBlob"

| where TimeGenerated > ago(24h)  
  
  
Azure Key Vault  
  
// List out Secrets

AzureDiagnostics

| where ResourceProvider == "MICROSOFT.KEYVAULT"

| where OperationName == "SecretList"

// Attempt to view passwords that don't exist

AzureDiagnostics

| where ResourceProvider == "MICROSOFT.KEYVAULT"

| where OperationName == "SecretGet"

| where ResultSignature == "Not Found"

// Viewing an actual existing password

AzureDiagnostics

| where ResourceProvider == "MICROSOFT.KEYVAULT"

| where OperationName == "SecretGet"

| where ResultSignature == "OK"

// Viewing a specific existing password

let CRITICAL\_PASSWORD\_NAME = "Tenant-Global-Admin-Password";

AzureDiagnostics

| where ResourceProvider == "MICROSOFT.KEYVAULT"

| where OperationName == "SecretGet"

| where id\_s contains CRITICAL\_PASSWORD\_NAME

// Updating a password Success

AzureDiagnostics

| where ResourceProvider == "MICROSOFT.KEYVAULT"

| where OperationName == "SecretSet"

// Updating a specific existing password Success

let CRITICAL\_PASSWORD\_NAME = "Tenant-Global-Admin-Password";

AzureDiagnostics

| where ResourceProvider == "MICROSOFT.KEYVAULT"

| where OperationName == "SecretSet"

| where id\_s endswith CRITICAL\_PASSWORD\_NAME

| where TimeGenerated > ago(2h)

// Failed access attempts

AzureDiagnostics

| where ResourceProvider == "MICROSOFT.KEYVAULT"

| where ResultSignature == "Unauthorized"

// Updating a specific existing secret in Key Vault

let CRITICAL\_PASSWORD\_NAME = "Tenant-Global-Admin-Password";

AzureDiagnostics

| where ResourceProvider == "MICROSOFT.KEYVAULT"

| where OperationName == "SecretSet"

| where id\_s endswith CRITICAL\_PASSWORD\_NAME  
  
  
Network Security Groups  
  
// Allowed inbound malicious flows

AzureNetworkAnalytics\_CL

| where FlowType\_s == "MaliciousFlow" and AllowedInFlows\_d >= 1

| project TimeGenerated, FlowType = FlowType\_s, IpAddress = SrcIP\_s, DestinationIpAddress = DestIP\_s, DestinationPort = DestPort\_d, Protocol = L7Protocol\_s, NSGRuleMatched = NSGRules\_s, InboundFlowCount = AllowedInFlows\_d