KQL Queries for Threat Hunting

Table of Contents

[KQL Basics 4](#_Toc166142627)

[Initial Access 4](#_Toc166142628)

[T1566 - Spamhaus 10 Most Abused TLDs 4](#_Toc166142629)

[Execution 5](#_Toc166142630)

[T1059.001 – Wscript to VBS file invoking PowerShell 5](#_Toc166142631)

[T1204.001 – Endpoints accessing .zip or .mov websites 6](#_Toc166142632)

[T1623 – MOVEit exploit hunting 7](#_Toc166142633)

[T1059.001 – PowerShell Base64 encoding 7](#_Toc166142634)

[T1059.001 – Changing PowerShell execution policy to insecure level 7](#_Toc166142635)

[T1059.003 – CVE-2023-36884 WinRAR spawning CMD 8](#_Toc166142636)

[Privilege escalation 8](#_Toc166142637)

[T1055.012 – OneNote spawning suspicious processes 8](#_Toc166142638)

[T1546.002 – Screensaver file invoking internet access 9](#_Toc166142639)

[T1623 – Screensaver file invoking suspicious processes 9](#_Toc166142640)

[Defense Evasion 9](#_Toc166142641)

[T1211 – CVE-2023-36884 Dropped file hunting 9](#_Toc166142642)

[T1218.005 – PowerShell Spawning MSHTA & initiating remote connection 10](#_Toc166142643)

[Compromised Credentials 10](#_Toc166142644)

[T1649 – Binaries using AnyDesk Compromised Certificate 10](#_Toc166142645)

[Discovery 10](#_Toc166142646)

[T1614 – Remcos RAT checking for geolocation through web 11](#_Toc166142647)

[T1087 – Possible SOAPHound Tool execution using specific arguments 11](#_Toc166142648)

[Command and Control 12](#_Toc166142649)

[T1071.001 – CVE-2023-36884 URL marker 12](#_Toc166142650)

[T1071.004 – DNS requests to suspicious TLDs 12](#_Toc166142651)

[Spamhaus 10 most Abused TLDs 12](#_Toc166142652)

[T1562.004 - Network Shell command for firewall to allow incoming RDP connections 13](#_Toc166142653)

[T1571 - RDP default listening port modification 14](#_Toc166142654)

[T1112 - RDP enable by modifying registry key 14](#_Toc166142655)

[Threat Detection 15](#_Toc166142656)

[Delivered emails identified as suspicious 15](#_Toc166142657)

[Detect inbound email domains from text list 15](#_Toc166142658)

[Detect inbound email domains from text list 16](#_Toc166142659)

[Detect lumma stealer using TeslaBrowser user agent 16](#_Toc166142660)

[OneNote invoking browser that produced smart screen warning 17](#_Toc166142661)

[Detect RaspBerry Robin malware cmd invoking msiexec 17](#_Toc166142662)

[Review recently received emails with attachments 18](#_Toc166142663)

[Review recently received emails with phishing related subject keywords 18](#_Toc166142664)

[Review recent UrlClick events 19](#_Toc166142665)

[Detect malware communication using SSL inspection 19](#_Toc166142666)

[References 20](#_Toc166142667)

# KQL Basics

Choose appropriate table

Data is organized into a hierarchy of databases, tables and columns, similar to SQL. For example, the DeviceNetworkEvents table in the advanced hunting schema contains information about network connections and related events.

where filters on a specific predicate

DeviceNetworkEvents

| where LocalIP == "192.168.0.1"

contains/has

Contains: Looks for any substring match

Has: Looks for a specific word (better performance)

DeviceNetworkEvents

| where DeviceName has "ComputerName"

ago

Returns the time offset relative to the time the query executes

DeviceNetworkEvents

| where Timestamp > ago(1d)

project

Selects the columns to include in the order specified

DeviceNetworkEvents

| where Timestamp > ago(1d)

| where DeviceName has "ComputerName"

| project Timestamp, ActionType, RemoteIP, RemotePort, RemoteUrl

# Initial Access

## T1566 - Spamhaus 10 Most Abused TLDs

**Description**

The following query will hunt for inbound emails with SenderMailFromDomain and SenderFromDomain that match Spamhaus 10 Most Abused Top Level Domains.

**Microsoft Defender XDR**

let SpamhausTLD = externaldata(TLD: string)[@"https://raw.githubusercontent.com/cyb3rmik3/Hunting-Lists/main/spamhaus-abused-tlds.csv"] with (format="csv", ignoreFirstRecord=True);

let Timeframe = 1d; // Choose the best timeframe for your investigation

let SMFDEvents = EmailEvents

| where Timestamp > ago(Timeframe)

| where EmailDirection == "Inbound"

| where SenderMailFromDomain has\_any (SpamhausTLD)

| project Timestamp, NetworkMessageId, SenderMailFromAddress, SenderFromAddress, SenderIPv4, RecipientEmailAddress, DeliveryAction, ThreatTypes, EmailAction, EmailActionPolicy, UserLevelAction, UserLevelPolicy, LatestDeliveryLocation, LatestDeliveryAction;

let SFDEvents = EmailEvents

| where Timestamp > ago(Timeframe)

| where EmailDirection == "Inbound"

| where SenderFromDomain has\_any (SpamhausTLD)

| project Timestamp, NetworkMessageId, SenderMailFromAddress, SenderFromAddress, SenderIPv4, RecipientEmailAddress, DeliveryAction, ThreatTypes, EmailAction, EmailActionPolicy, UserLevelAction, UserLevelPolicy, LatestDeliveryLocation, LatestDeliveryAction;

(union isfuzzy=true

(SMFDEvents),

(SFDEvents)

| summarize SenderMailFromAddresses = make\_set(SenderMailFromAddress), SenderFromAddresses = make\_set(SenderFromAddress), TimeReceived = arg\_max(Timestamp, \*) by NetworkMessageId

| project-reorder TimeReceived, NetworkMessageId, SenderMailFromAddress, SenderFromAddress, SenderIPv4, RecipientEmailAddress, DeliveryAction, ThreatTypes, EmailAction, EmailActionPolicy, UserLevelAction, UserLevelPolicy, LatestDeliveryLocation, LatestDeliveryAction

| sort by TimeReceived

)

# Execution

## T1059.001 – Wscript to VBS file invoking PowerShell

**Description**

This hunting query is based on a GULOADER payload delivered through a .vbs file which invoked PowerShell to gain foothold on the device.

**Microsoft 365 Defender**

DeviceProcessEvents

// Define the time you are interested to look into

| where Timestamp > ago(1d)

| where InitiatingProcessParentFileName contains @"wscript.exe"

// Command line includes VBS file execution

| where InitiatingProcessCommandLine contains ".vbs"

// Invoking PowerShell or Command Line

| where InitiatingProcessFileName has\_any (@"powershell.exe", @"pwsh.exe", @"powershell\_ise.exe", @"cmd.exe")

// Define elements that should be available in the results

| project Timestamp, DeviceName, AccountName, InitiatingProcessFileName, InitiatingProcessCommandLine, InitiatingProcessParentFileName

**Microsoft Sentinel**

DeviceProcessEvents

// Define the time you are interested to look into

| where TimeGenerated > ago(1d)

| where InitiatingProcessParentFileName contains @"wscript.exe"

// Command line includes VBS file execution

| where InitiatingProcessCommandLine contains ".vbs"

// Invoking PowerShell or Command Line

| where InitiatingProcessFileName has\_any (@"powershell.exe", @"pwsh.exe", @"powershell\_ise.exe", @"cmd.exe")

// Define elements that should be available in the results

| project Timestamp, DeviceName, AccountName, InitiatingProcessFileName, InitiatingProcessCommandLine, InitiatingProcessParentFileName

## T1204.001 – Endpoints accessing .zip or .mov websites

**Description**

Google recently announced a series of new gTLDs available for registration amongst them, .zip and .mov. While Google saw an opportunity in providing public access to these domains, defenders are already worried that they could be used for malicious purposes. The following queries  provides a hunting opportunity for .zip and .mov accessed domains originating from file explorer.

**Microsoft 365 Defender**

DeviceNetworkEvents

// Define the time you are interested to look into

| where Timestamp > ago(1d)

// Remove the line below in case you want to look into both successful and unsuccessful events

| where ActionType == "ConnectionSuccess"

// The line below refers to connections made when requesting a .zip through file explorer

// | where InitiatingProcessFileName == @"svchost.exe"

// Define RemoteURL

| where RemoteUrl startswith "https://" or RemoteUrl startswith "http://"

// Define domain as a string that includes a domain exclusively leaving outside .zip or .mov accessed files for download

| extend domain = tostring(extract("https?://([^:/]\*)(:?)(/|$)", 1, RemoteUrl))

// String should exclusively look for .zip or .mov TLDs

| where domain endswith ".zip" or domain endswith ".mov"

| project Timestamp, DeviceName, ActionType, RemoteUrl

// Sort by newsest events first

| sort by Timestamp desc

**Microsoft Sentinel**

DeviceNetworkEvents

// Define the time you are interested to look into

| where TimeGenerated > ago(1d)

// Remove the line below in case you want to look into both successful and unsuccessful events

| where ActionType == "ConnectionSuccess"

// The line below refers to connections made when requesting a .zip through file explorer

// | where InitiatingProcessFileName == @"svchost.exe"

// Define RemoteURL

| where RemoteUrl startswith "https://" or RemoteUrl startswith "http://"

// Define domain as a string that includes a domain exclusively leaving outside .zip or .mov accessed files for download

| extend domain = tostring(extract("https?://([^:/]\*)(:?)(/|$)", 1, RemoteUrl))

// String should exclusively look for .zip or .mov TLDs

| where domain endswith ".zip" or domain endswith ".mov"

| project Timestamp, DeviceName, ActionType, RemoteUrl

// Sort by newsest events first

| sort by Timestamp desc

## T1623 – MOVEit exploit hunting

**Description**

Based on the threat reports presented with regards to MOVEit vulnerability CVE-2023-34362 w3wp.exe spawns csc.exe in order to compile a malicious .dll file.

**Microsoft 365 Defender & Microsoft Sentinel**

DeviceProcessEvents

// Optionally, you can narrow down your hunt based on your MOVEit hosts

// | where DeviceName has "DeviceNameHere"

| where ActionType == "ProcessCreated"

| where InitiatingProcessParentFileName has "w3wp.exe"

| where InitiatingProcessFileName has "csc.exe"

| where InitiatingProcessCommandLine has "moveitdmz pool"

## T1059.001 – PowerShell Base64 encoding

**Description**

Hunting for Base64 encoded command lines in PowerShell is crucial to detect and mitigate potential cyber threats. While the analytic below will detect Base64 encoded commands, a fine tuning is required for your environment.

**Microsoft 365 Defender & Microsoft Sentinel**

DeviceProcessEvents

// Define timeframe

| where Timestamp > ago(1d)

| where FileName has\_any (@"powershell.exe", @"pwsh.exe", @"powershell\_ise.exe")

| where ProcessCommandLine contains "base64"

| summarize arg\_max(Timestamp, \*) by DeviceName

## T1059.001 – Changing PowerShell execution policy to insecure level

**Description**

PowerShell's execution policy is a safety feature that controls the conditions under which PowerShell loads configuration files and runs scripts. This query will help you identify execution policy changes. Also, you may fine tune the query by excluding InitiatingProcessFileName and InitiatingProcessParentFileName from your environment's applications.

**Microsoft XDR**

let Timeframe = 7d; // Choose the best timeframe for your investigation

let cmdlet = dynamic([@'-executionpolicy ', @' -ep ', @' -exec ']);

let parameters = dynamic([@'Bypass ', @'Unrestricted']);

let exinitapps = datatable(excludedinitapps :string) // Add as many initiating process filenames you would like to exclude

["applicationfilename1.exe",

"applicationfilename2.exe",

"applicationfilename3.exe"];

let exparinitapps = datatable(excludedparinitapps :string) // Add as many initiating parent process filenames you would like to exclude

["applicationfilename1.exe",

"applicationfilename2.exe",

"applicationfilename3.exe"];

DeviceProcessEvents

| where Timestamp > ago(Timeframe)

| where ProcessCommandLine has\_any(cmdlet) and ProcessCommandLine has\_any(parameters)

| where not(InitiatingProcessFileName in (['exinitapps']))

| where not(InitiatingProcessParentFileName in (['exparinitapps']))

| sort by Timestamp desc

## T1059.003 – CVE-2023-36884 WinRAR spawning CMD

**Description**

If you are using a RARLabs WinRAR version prior to 6.23, you are vulnerable to CVE-2023-36884 which allows RCE. A benign file like PDF or JPG could facilitate the execution of arbitrary code.

**Microsoft 365 Defender & Microsoft Sentinel**

DeviceProcessEvents

| where InitiatingProcessParentFileName has @"winrar.exe"

| where InitiatingProcessFileName has @"cmd.exe"

| project Timestamp, DeviceName, FileName, FolderPath, ProcessCommandLine, AccountName

| sort by Timestamp desc

# Privilege escalation

|  |
| --- |

## T1055.012 – OneNote spawning suspicious processes

**Description**

This hunting query is based on a GULOADER payload delivered through a .vbs file which invoked PowerShell to gain foothold on the device.

**Microsoft 365 Defender**

DeviceProcessEvents

// Define the time you are interested to look into

| where Timestamp > ago(1d)

| where InitiatingProcessParentFileName contains @"wscript.exe"

// Command line includes VBS file execution

| where InitiatingProcessCommandLine contains ".vbs"

// Invoking PowerShell or Command Line

| where InitiatingProcessFileName has\_any (@"powershell.exe", @"pwsh.exe", @"powershell\_ise.exe", @"cmd.exe")

// Define elements that should be available in the results

| project Timestamp, DeviceName, AccountName, InitiatingProcessFileName, InitiatingProcessCommandLine, InitiatingProcessParentFileName

**Microsoft Sentinel**

DeviceProcessEvents

// Define the time you are interested to look into

| where TimeGenerated > ago(1d)

| where InitiatingProcessParentFileName contains @"wscript.exe"

// Command line includes VBS file execution

| where InitiatingProcessCommandLine contains ".vbs"

// Invoking PowerShell or Command Line

| where InitiatingProcessFileName has\_any (@"powershell.exe", @"pwsh.exe", @"powershell\_ise.exe", @"cmd.exe")

// Define elements that should be available in the results

| project Timestamp, DeviceName, AccountName, InitiatingProcessFileName, InitiatingProcessCommandLine, InitiatingProcessParentFileName

## T1546.002 – Screensaver file invoking internet access

**Description**

This hunting query is based on a RedLine stealer malware delivered through a .scr file which invoked a process accessing the internet to deliver payload.

**Microsoft 365 Defender**

DeviceNetworkEvents

| where Protocol contains "tcp"

| where RemoteIPType contains "Public"

| where InitiatingProcessFileName contains ".scr"

| summarize arg\_max(Timestamp, \*) by DeviceName

| project Timestamp, DeviceName

## T1623 – Screensaver file invoking suspicious processes

**Description**

This hunting query is based on a RedLine stealer malware delivered through a .scr file which invoked a suspicious processes including command prompt, powershell or any other .exe file.

**Microsoft 365 Defender**

DeviceProcessEvents

| where FileName has\_any (@"cmd.exe", @"powershell.exe", @".exe")

| where InitiatingProcessFileName contains ".scr"

| summarize arg\_max(Timestamp, \*) by DeviceName

| project Timestamp, DeviceName

# Defense Evasion

## T1211 – CVE-2023-36884 Dropped file hunting

**Description**

Following relevant puplic reports of analysis with regards to CVE-2023-36884 exploitation, the following query can help hunt for the second stage dropped file.

**Microsoft 365 Defender & Microsoft Sentinel**

DeviceFileEvents

| where ActionType == "FileCreated"

| where FolderPath startswith @"C:\Users\"

| where FolderPath contains @"\AppData\Roaming\Microsoft\Office\Recent\"

| where FolderPath endswith @"\file001.url"

## T1218.005 – PowerShell Spawning MSHTA & initiating remote connection

**Description**

MSHTA is heavily used by adversaries as it allows to execute arbitrary code using a Windows native application. A scenario observed includes powershell.exe spawning mshta.exe which by itself is suspicious but it is almost certainly malicious when a remote call initiates through an .hta file. The following query will track powershell spawning mshta initiating a remote connection.

**Microsoft 365 Defender & Microsoft Sentinel**

let Process = DeviceProcessEvents

| where InitiatingProcessParentFileName has\_any (@"powershell.exe", @"pwsh.exe", @"powershell\_ise.exe")

| where InitiatingProcessFileName has @"mshta.exe"

| project Timestamp, DeviceName, AccountDomain, AccountName;

Process

| join (DeviceNetworkEvents

| where RemoteIP !has "" or RemoteUrl !has ""

| project DeviceName, RemoteIP, RemoteUrl

) on DeviceName

# Compromised Credentials

## T1649 – Binaries using AnyDesk Compromised Certificate

**Description**

The following query will hunt for binaries not related to AnyDesk, signed with a potentially compromised signing certificate of AnyDesk.

**Microsoft Defender XDR**

let Timeframe = 7d; // Choose the best timeframe for your investigation

let SuspiciousAnydeskFileCertificate = DeviceFileCertificateInfo

| where Timestamp > ago(Timeframe)

| where CertificateSerialNumber =~ "0dbf152deaf0b981a8a938d53f769db8" // Compromised Certificate Serial Number

| where Issuer == "DigiCert Trusted G4 Code Signing RSA4096 SHA384 2021 CA1"

| project Timestamp, DeviceName, SHA1;

SuspiciousAnydeskFileCertificate

| join (DeviceProcessEvents

| where Timestamp > ago(Timeframe)

| where ProcessVersionInfoCompanyName !contains @"AnyDesk"

| project SHA1, ActionType, FileName, FolderPath, ProcessVersionInfoCompanyName, ProcessVersionInfoProductName, ProcessCommandLine, AccountName, InitiatingProcessAccountName, InitiatingProcessFileName, InitiatingProcessCommandLine

)on SHA1

| sort by Timestamp desc

# Discovery

## T1614 – Remcos RAT checking for geolocation through web

**Description**

SANS ISC published a diary on 30/05/2023 where ModiLoader installs a Remcos RAT payload which checks for geolocation through web by geoplugin[.]net. The following query checks whether an attempt to connect to geoplugin[.]net has been made by a non-browser application. Query can be modified based on your environment and the browsers used, also other geolocation services could be checked.

**Microsoft 365 Defender**

// Define browser executable filenames

let Browser = dynamic(["firefox.exe", "msedge.exe", "chrome.exe", "opera.exe", "brave.exe"]);

DeviceNetworkEvents

// Define timeframe

| where Timestamp > ago(30d)

| where not(InitiatingProcessFileName in (['Browser']))

// Define service used to locate geographical information

| where RemoteUrl contains 'geoplugin.net'

| project Timestamp, DeviceName, LocalIP, RemoteUrl, RemoteIP, RemotePort, InitiatingProcessFileName, InitiatingProcessAccountUpn

| sort by Timestamp desc

**Microsoft Sentinel**

// Define browser executable filenames

let Browser = dynamic(["firefox.exe", "msedge.exe", "chrome.exe", "opera.exe", "brave.exe"]);

DeviceNetworkEvents

// Define timeframe

| where TimeGenerated > ago(30d)

| where not(InitiatingProcessFileName in (['Browser']))

// Define service used to locate geographical information

| where RemoteUrl contains 'geoplugin.net'

| project Timestamp, DeviceName, LocalIP, RemoteUrl, RemoteIP, RemotePort, InitiatingProcessFileName, InitiatingProcessAccountUpn

| sort by Timestamp desc

## T1087 – Possible SOAPHound Tool execution using specific arguments

**Description**

SOAPHound is a custom-developed .NET data collector tool which can be used to enumerate Active Directory environments via the Active Directory Web Services (ADWS) protocol. The following query will detect possible SOAPHound activity, based on the execution options, and relevant arguments.

**Microsoft Defender XDR & Sentinel**

DeviceProcessEvents

| where ProcessCommandLine has\_any (" --buildcache "," --bhdump ", " --certdump "," --dnsdump ")

and ProcessCommandLine has\_any (" -c "," --cachefilename ", " -o "," --outputdirectory")

# Command and Control

## T1071.001 – CVE-2023-36884 URL marker

**Description**

Following relevant puplic reports of analysis with regards to CVE-2023-36884 exploitation, the following query can help hunt a unique URL marker identified.

**Microsoft 365 Defender & Microsoft Sentinel**

DeviceNetworkEvents

| where ActionType == "HttpConnectionInspected"

| extend json = todynamic(AdditionalFields)

| extend method = tostring(json.method), uri = tostring(json.uri), direction = tostring(json.direction)

| where method == "GET"

| where uri contains "/MSHTML\_C7/"

| where direction == 'Out'

## T1071.004 – DNS requests to suspicious TLDs

**Description**

Zeek network layer signals for MDE includes the DnsConnectionInspected table which provides fruitful information about DNS connections. By taking into account netcraft's top 20 TLDs that are being used for cybercrime, you can hunt for suspicious DNS requests.

**Microsoft 365 Defender & Microsoft Sentinel**

let SuspiciousTLD = externaldata(TLD: string)[@"https://raw.githubusercontent.com/cyb3rmik3/Hunting-Lists/main/netcraft-tlds.csv"] with (format="csv", ignoreFirstRecord=True);

DeviceNetworkEvents

| where ActionType == "DnsConnectionInspected"

| extend AdditionalFields = todynamic(AdditionalFields)

| extend DnsQuery = tostring(AdditionalFields.query), ResponseCode = tostring(AdditionalFields.rcode\_name), Direction = tostring(AdditionalFields.direction)

| extend TLDArray = split(DnsQuery,'.')

| extend TLD = strcat(".",TLDArray[array\_length(TLDArray)-1])

| where Direction == "Out"

| project DeviceName, DnsQuery, ResponseCode, TLDArray, TLD

| join SuspiciousTLD on $left.TLD == $right.TLD

# Spamhaus 10 most Abused TLDs

**Description**

The following query will hunt for inbound emails with SenderMailFromDomain and SenderFromDomain that match Spamhaus 10 Most Abused Top Level Domains.

**Microsoft Defender XDR**

let SpamhausTLD = externaldata(TLD: string)[@"https://raw.githubusercontent.com/cyb3rmik3/Hunting-Lists/main/spamhaus-abused-tlds.csv"] with (format="csv", ignoreFirstRecord=True);

let Timeframe = 1d; // Choose the best timeframe for your investigation

let SMFDEvents = EmailEvents

| where Timestamp > ago(Timeframe)

| where EmailDirection == "Inbound"

| where SenderMailFromDomain has\_any (SpamhausTLD)

| project Timestamp, NetworkMessageId, SenderMailFromAddress, SenderFromAddress, SenderIPv4, RecipientEmailAddress, DeliveryAction, ThreatTypes, EmailAction, EmailActionPolicy, UserLevelAction, UserLevelPolicy, LatestDeliveryLocation, LatestDeliveryAction;

let SFDEvents = EmailEvents

| where Timestamp > ago(Timeframe)

| where EmailDirection == "Inbound"

| where SenderFromDomain has\_any (SpamhausTLD)

| project Timestamp, NetworkMessageId, SenderMailFromAddress, SenderFromAddress, SenderIPv4, RecipientEmailAddress, DeliveryAction, ThreatTypes, EmailAction, EmailActionPolicy, UserLevelAction, UserLevelPolicy, LatestDeliveryLocation, LatestDeliveryAction;

(union isfuzzy=true

(SMFDEvents),

(SFDEvents)

| summarize SenderMailFromAddresses = make\_set(SenderMailFromAddress), SenderFromAddresses = make\_set(SenderFromAddress), TimeReceived = arg\_max(Timestamp, \*) by NetworkMessageId

| project-reorder TimeReceived, NetworkMessageId, SenderMailFromAddress, SenderFromAddress, SenderIPv4, RecipientEmailAddress, DeliveryAction, ThreatTypes, EmailAction, EmailActionPolicy, UserLevelAction, UserLevelPolicy, LatestDeliveryLocation, LatestDeliveryAction

| sort by TimeReceived

)

# T1562.004 - Network Shell command for firewall to allow incoming RDP connections

**Description**

Attackers can attempt to enable RDP, including leveraging multiple living-off-the-land tools. Once RDP is enabled, it allows the attackers to use any number of dual-use tools that leverage the RDP protocol. The following query will hunt for an attempt to create a firewall rule to specifically allow all incoming RDP connections using a Network Shell (netsh) command.

**Microsoft Defender XDR**

let Timeframe = 1d; // Choose the best timeframe for your investigation

let fwcommands = dynamic([@"advfirewall", @"firewall", @"add rule", @"dir=in", @"localport=3389", @"action=allow"]);

DeviceProcessEvents

| where Timestamp > ago(Timeframe)

| where FileName has @"netsh.exe"

| where ProcessCommandLine has\_all (fwcommands)

| where ActionType has "ProcessCreated"

| project DeviceId, DeviceName, ProcessCommandLine

# T1571 - RDP default listening port modification

**Description**

Changing the default port 3389 to a non-standard port, could indicate a potential APT behaviour to avoid being detected abusing RDP connection.

**Microsoft Defender XDR**

let Timeframe = 1d; // Choose the best timeframe for your investigation

DeviceRegistryEvents

| where Timestamp > ago(Timeframe)

| where RegistryKey == @"HKEY\_LOCAL\_MACHINE\SYSTEM\ControlSet001\Control\Terminal Server\WinStations\RDP-Tcp"

| where RegistryValueName == @"PortNumber"

| where RegistryValueData != @"3389"

| where ActionType == @"RegistryValueSet"

| project Timestamp, DeviceName, PreviousRegistryValueName, PreviousRegistryValueData, InitiatingProcessFileName

# T1112 - RDP enable by modifying registry key

**Description**

Attackers can attempt to enable RDP, including leveraging multiple living-off-the-land tools. Once RDP is enabled, it allows the attackers to use any number of dual-use tools that leverage the RDP protocol. The following query will hunt for an attempt to enable RDP by simply modifying a registry key.

**Microsoft Defender XDR**

let rdpcommands = dynamic([@"fDenyTSConnections", @"REG\_DWORD /d 0"]);

DeviceProcessEvents

| where FileName has @"reg.exe"

| where ProcessCommandLine has\_all (rdpcommands)

| project DeviceId, DeviceName, ProcessCommandLine, Start = Timestamp

| join kind = inner (DeviceRegistryEvents

| where RegistryKey == @"HKEY\_LOCAL\_MACHINE\SYSTEM\ControlSet001\Control\Terminal Server"

| where RegistryValueName == @"fDenyTSConnections"

| where ActionType == @"RegistryValueSet"

| where RegistryValueData == @"0"

| where InitiatingProcessFileName == @"reg.exe"

| project DeviceId, End = Timestamp)

on DeviceId

| where (End - Start) between (0min .. 1min)

| project Start, DeviceId, DeviceName, ProcessCommandLine

# Threat Detection

# Delivered emails identified as suspicious

**Description**

The following query will present email details that have been identified as suspicious after delivery.

**Microsoft 365 Defender**

let CompromizedEmailAddress = ""; // Insert the email address of the compromised email address

let Timeframe = 2d; // Choose the best timeframe for your investigation

let EmailInformation = EmailEvents

| where RecipientEmailAddress == CompromizedEmailAddress

| where Timestamp > ago(Timeframe)

| where DeliveryAction != "Blocked"

| project Timestamp, NetworkMessageId, SenderMailFromAddress, SenderFromAddress, SenderDisplayName, ThreatNames;

EmailInformation

| join (EmailPostDeliveryEvents

| where ThreatTypes != ""

| project Timestamp, NetworkMessageId, Action, ActionType, ActionTrigger, ActionResult, DeliveryLocation, ThreatTypes, DetectionMethods

) on NetworkMessageId

| sort by Timestamp desc

# Detect inbound email domains from text list

**Description**

Following a challenging request from a fellow Microsoft Tech Community member, the query below will help you detect inbound emails which match the domains provided from a public text file. It also allows to exclude specific domains, if required.

**Microsoft 365 Defender**

let domainList = externaldata(domain: string) [@"https://raw.githubusercontent.com/tsirolnik/spam-domains-list/master/spamdomains.txt"] with (format="txt"); // Change the text file to whatever you want

let excludedDomains = datatable(excludeddomain :string) // Add as many domains you would like to exclude

["domain1.tld",

"domain2.tld",

"domain3.tld"];

let Timeframe = 1d; // Choose the best timeframe for your investigation

let SuspiciousEmails = EmailEvents

| where Timestamp > ago(Timeframe)

| where EmailDirection == "Inbound"

| extend EmailDomain = tostring(split(SenderMailFromAddress, '@')[1])

| join kind=inner (domainList) on $left.EmailDomain == $right.domain

| where not(EmailDomain in (['excludedDomains']))

| project Timestamp, NetworkMessageId, SenderMailFromAddress, SenderFromAddress, SenderDisplayName, RecipientEmailAddress, EmailDomain, domain, Subject, LatestDeliveryAction;

SuspiciousEmails

| join (EmailEvents

| project NetworkMessageId

)on NetworkMessageId

| sort by Timestamp desc

# Detect inbound email domains from text list

**Description**

Recent rise of Remote Monitoring and Management (RMM) tools used by prominent Threat Actors for lateral movement and command and control (C2) has led to significantly getting worried about the use of legitimate software such as Teamviewer, NetSupport Manager etc. The following query has been crafted to utilize the ProcessVersionInfoCompanyName table with a hunting list created by installing and testing corresponding tools.

**Microsoft 365 Defender & Microsoft Sentinel**

let RMMSoftware = externaldata(RMMSoftware: string)[@"https://raw.githubusercontent.com/cyb3rmik3/Hunting-Lists/main/rmm-software.csv"] with (format="csv", ignoreFirstRecord=True);

let ExclDevices = datatable(excludeddev :string) // Add as many devices you would like to exclude

["DeviceName1",

"DeviceName2",

"DeviceName3"];

let Timeframe = 7d; // Choose the best timeframe for your investigation

DeviceProcessEvents

| where Timestamp > ago(Timeframe)

| where ProcessVersionInfoCompanyName has\_any (RMMSoftware)

| where not(DeviceName in (['ExclDevices']))

| project Timestamp, DeviceName, ActionType, FileName, FolderPath, ProcessVersionInfoCompanyName, ProcessVersionInfoProductName, ProcessCommandLine, AccountName, InitiatingProcessAccountName, InitiatingProcessFileName, InitiatingProcessCommandLine

| sort by Timestamp desc

# Detect lumma stealer using TeslaBrowser user agent

**Description**

Recently seen in the wild rising further, Lumma stealer has been observed to perform HTTP GET method, while using “TeslaBrowser/5.5” user agent.

**Microsoft 365 Defender & Microsoft Sentinel**

DeviceNetworkEvents

| where ActionType == "HttpConnectionInspected"

| extend json = todynamic(AdditionalFields)

| extend direction = tostring(json.direction), method = tostring(json.method), user\_agent = tostring(json.user\_agent)

| where direction == "Out"

| where method == "GET"

| where user\_agent contains @"TeslaBrowser"

| project Timestamp, DeviceName, RemoteIP, RemotePort, RemoteUrl

| sort by Timestamp desc

# OneNote invoking browser that produced smart screen warning

**Description**

A detection rule for OneNote files, invoking browser (inline URL) which produced a smart screen URL warning.

**Microsoft 365 Defender & Microsoft Sentinel**

DeviceInfo

let Process = DeviceProcessEvents

| where InitiatingProcessFileName contains "onenote.exe"

// Define any other browser files below that may be present in your environment

| where FileName has\_any ("firefox.exe","msedge.exe","chrome.exe")

| project Timestamp, DeviceId, DeviceName, AccountDomain, AccountName;

// Joining DeviceEvents table to correlate SmartScreen URL warnings

Process

| join (DeviceEvents

| where ActionType == "SmartScreenUrlWarning"

| project DeviceId, DeviceName, InitiatingProcessAccountUpn, RemoteUrl

) on DeviceId

# Detect RaspBerry Robin malware cmd invoking msiexec

**Description**

A detection opportunity by taking into consideration the fact that RaspBerry Robin domains used have the following structure:

http://xx.xx:8080/xx/hostname?username

http://xx.xx:8080/xx/hostname=username

http://xx.xx:8080/xx/hostname

The query also incorporates the fact that the parent process is a command prompt and

**Microsoft 365 Defender & Microsoft Sentinel**

let rbr01 = @'[A-Za-z0-9]+://[A-Za-z0-9]+\.[A-Za-z0-9]+:8080/[A-Za-z0-9]+/[A-Za-z0-9]+\?[A-Za-z0-9]+';

let rbr02 = @'[A-Za-z0-9]+://[A-Za-z0-9]+\.[A-Za-z0-9]+:8080/[A-Za-z0-9]+/[A-Za-z0-9]+\=[A-Za-z0-9]+';

let rbr03 = @'[A-Za-z0-9]+://[A-Za-z0-9]+\.[A-Za-z0-9]+:8080/[A-Za-z0-9]+/[A-Za-z0-9]+';

DeviceProcessEvents

| where Timestamp > ago(1d)

| where InitiatingProcessFileName has @'cmd.exe'

| where ProcessCommandLine has "msiexec"

| where ProcessCommandLine matches regex rbr01 or ProcessCommandLine matches regex rbr02 or ProcessCommandLine matches regex rbr03

| project Timestamp, DeviceName, AccountName, FileName, InitiatingProcessFileName, ProcessCommandLine

# Review recently received emails with attachments

**Description**

The following query will list all emails received on the Timeframe specified that haven’t been blocked and have an attachment. This could help get an overview of the email attachments recently received that might rise suspicions.

**Microsoft 365 Defender**

let CompromizedEmailAddress = ""; // Insert the email address of the compromised email address

let Timeframe = 2d; // Choose the best timeframe for your investigation

let EmailInformation = EmailEvents

| where RecipientEmailAddress == CompromizedEmailAddress

| where Timestamp > ago(Timeframe)

| where DeliveryAction != "Blocked"

| where AttachmentCount != "0"

| project Timestamp, NetworkMessageId, SenderMailFromAddress, SenderFromAddress, SenderDisplayName, ThreatNames;

EmailInformation

| join (EmailAttachmentInfo

| project NetworkMessageId, FileName, FileType, FileSize

) on NetworkMessageId

| sort by Timestamp desc

# Review recently received emails with phishing related subject keywords

**Description**

The following query will go through a set of keywords included in a curated list that could be found in email’s subject, that could potentially be correlated to phishing emails. This query will most probably return a lot of false/positives, however it could potentially return results significant enough to go through.

**Microsoft 365 Defender**

let CompromizedEmailAddress = ""; // Insert the email address of the compromised email address

let SuspiciousKeywords = externaldata(Keywords: string)[@"https://raw.githubusercontent.com/cyb3rmik3/Hunting-Lists/main/phishing-keywords.csv"] with (format="csv", ignoreFirstRecord=True);

let Timeframe = 2d; // Choose the best timeframe for your investigation

EmailEvents

| where RecipientEmailAddress == CompromizedEmailAddress

| where Timestamp > ago(Timeframe)

| where Subject has\_any (SuspiciousKeywords)

| where DeliveryAction == "Delivered"

| project Timestamp, SenderMailFromAddress, SenderFromAddress, SenderDisplayName, SenderMailFromDomain, SenderFromDomain, SenderIPv4, AttachmentCount, UrlCount, LatestDeliveryAction

| sort by Timestamp desc

# Review recent UrlClick events

**Description**

The following query will help identify emails with URLs inline, where the user took action and clicked any of them and the URL wasn’t blocked.

**Microsoft 365 Defender**

let CompromizedEmailAddress = ""; // Insert the email address of the compromised email address

let Timeframe = 2d; // Choose the best timeframe for your investigation

let EmailInformation = EmailEvents

| where RecipientEmailAddress == CompromizedEmailAddress

| where Timestamp > ago(Timeframe)

| where UrlCount != "0"

| project Timestamp, NetworkMessageId, SenderMailFromAddress, SenderFromAddress, SenderDisplayName, ThreatNames;

EmailInformation

| join (UrlClickEvents

| where ActionType != "ClickBlocked"

| where Workload == "Email"

| project Timestamp, Url, IPAddress, NetworkMessageId

) on NetworkMessageId

| sort by Timestamp desc

# Detect malware communication using SSL inspection

**Description**

Recently updated Zeek network layer signals for MDE with SSL inspection offer new detection and hunting possibilities. As some malware use common, default or re-used certificates, the following query could help detect AsyncRAT, Cobalt Strike, QuasarRAT, Laplas Clipper, DcRAT, VenomRAT, BitRAT and Mythic C2.

**Microsoft 365 Defender & Microsoft Sentinel**

let IssuerContent = dynamic([@"Major Cobalt Strike", @"Laplas.app", @"Pwn3rs", @"operators"]);

let SubjectContent = dynamic([@"AsyncRAT Server", @"Major Cobalt Strike", @"Quasar Server CA", @"Laplas.app", @"Mythic", @"DcRat", @"VenomRAT", @"BitRAT", @"desas.digital", @"multiplayer" ]);

let Timeframe = 1d; // Choose the best timeframe for your investigation

DeviceNetworkEvents

| where Timestamp > ago(Timeframe)

| where ActionType == "SslConnectionInspected"

| extend AdditionalFields = todynamic(AdditionalFields)

| extend issuer = tostring(AdditionalFields.issuer), subject = tostring(AdditionalFields.subject), direction = tostring(AdditionalFields.direction)

| where direction == "Out" and not(ipv4\_is\_private(RemoteIP))

// Define issuer and subject parameters

| where issuer has\_any (IssuerContent) or subject has\_any (SubjectContent)

| sort by Timestamp desc

# References

<https://github.com/cyb3rmik3/KQL-threat-hunting-queries/>

<https://github.com/rod-trent/MustLearnKQL>

## Capture the flag and other training/learning

[KC7 Cyber](https://kc7cyber.com/) is a new way to learn cybersecurity that's hands-on, fun, and engaging.

[Kusto Detective Agency](https://detective.kusto.io/) is a set of challenges that is designed to help you learn the KQL.

https://github.com/cyb3rmik3/KQL-threat-hunting-queries/tree/main?tab=readme-ov-file#kql-training