Searching for persistence using Microsoft Defender for Endpoint

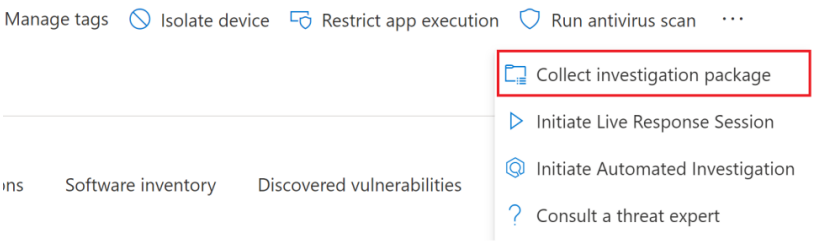
During incident response scenarios, one critical aspect is to look for persistence in endpoints, as attackers commonly setup persistence on one or more endpoints to gain command and control capabilities long after the investigators finish their work. The following scenario explains how to use Defender for Endpoint capabilities to hunt for this.

Defender for endpoint allows to find for persistence using hunting queries, several great examples can be found under the following link, but most of these take into consideration that the product is enabled prior to the attacker setting the persistence mechanism. If Defender for Endpoint is setup as part of the incident response, the search method can´t be used.

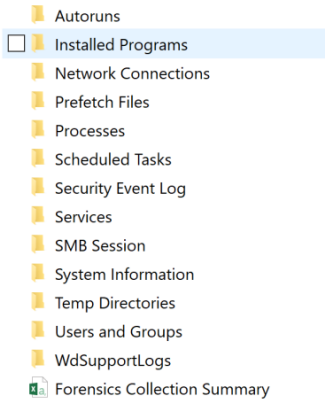
Microsoft-365-Defender-Hunting-Queries/Persistence at master · microsoft/Microsoft-365-Defender-Hunting-Queries · GitHub

Alternatively, an investigation package can be collected for a machine as explained here

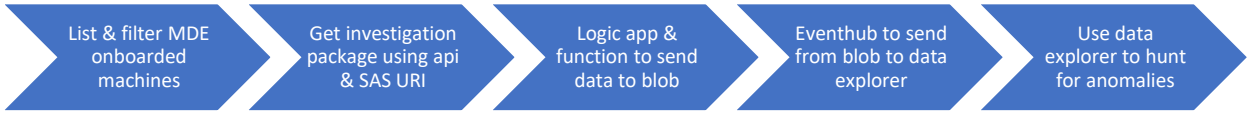
https://docs.microsoft.com/en-us/microsoft-365/security/defender-endpoint/respond-machine-alerts?view=o365-worldwide#collect-investigation-package-from-devices



The package contains this information at the time of writing.



We will collect this information at scale using the Defender for Endpoint API, send this data onto Azure Blob for long term retention and from there into an Azure Data Explorer cluster so we can hunt for persistence in installed programs, scheduled tasks, services, etc.



The following snippet of code, (this code is for demonstration purposes only,and offers no warranties) Get the list of the machines, contact the API to collect an investigation package, get a SAS URI to each investigation package and call a logic app passing the SAS URI as reference.

Collect investigation package API | Microsoft Docs

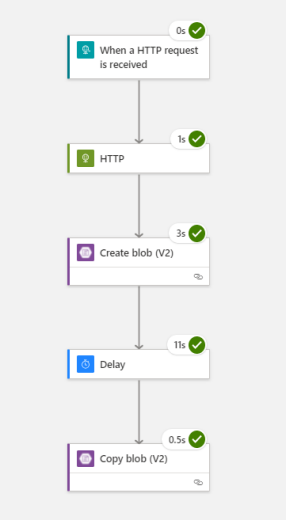
Get package SAS URI API | Microsoft Docs

```
#use app to query api, followed https://docs.microsoft.com/en-us/microsoft-
365/security/defender-endpoint/api-hello-world?view=o365-worldwide
$appSecret="<app secret>"
$appId ="<app id>"
$tenantId ="<tenant id>"
$subscription="<subscription id>"
$LogicAppsUrl="<logic app url>"

#get token into header
$resourceAppIdUri = 'https://api.securitycenter.microsoft.com'
$oAuthUri = "https://login.microsoftonline.com/$TenantId/oauth2/token"
$authBody = [Ordered] @{
     resource = "$resourceAppIdUri"
     client_id = "$appId"
     client_secret = "$appSecret"
     grant_type = 'client_credentials'
}
$authResponse = Invoke-RestMethod -Method Post -Uri $oAuthUri -Body $authBody -
ErrorAction Stop
$token = $authResponse.access_token
Out-File -FilePath "./Latest-token.txt" -InputObject $token
return $token
$headers = @{
    'Content-Type' = 'application/json'
    Accept = 'application/json'
    Authorization = "Bearer $token"
}
#Wait for token to be accepted by MDE
Start-Sleep -Seconds 30
#list machines
$machines=Invoke-RestMethod -Method Get -uri
"https://api.securitycenter.microsoft.com/api/machines" -Headers $headers

#json for the body of the automated investigation request
$body = @"
{
     "Comment": "Automated collection from ps"
}
"@
```

```
$packages=@{}
$machines= $machines | where {$_.value.osPlatform -eq "Windows10"}
#get over the list of machines and ask for investigation package, it only works on Win
10 over 1703
foreach ($machine in $machines.value){
    if ($machine.osPlatform -ne "Windows10"){
        break
        }
    $machineid=$machine.id
    $packagestatus=$null
    try{
        $packagestatus=Invoke-RestMethod -Method post -uri
"https://api.securitycenter.microsoft.com/api/machines/$machineid/collectInvestigation
Package" -Headers $headers -Body $body
        $packages.Add($machine.computerDnsName,$packagestatus)

        }
    catch
        {
        Write-output "failed to collect" $machine.computerDnsName
        }
#wait to acknowledge for api limits
 Start-Sleep -Seconds 30
    }
$SASURIs=@{}
#go over collected packages and get the SASURI, then call the logic app
foreach ($package in $packages.GetEnumerator()){
    $SASURI=$null
    $packageid=$package.value.id
    $SASURI=Invoke-RestMethod -Method get -uri
"https://api.securitycenter.microsoft.com/api/machineactions/$packageid/getPackageUri"
-Headers $headers
    $SASURIs.Add($package.Value.computerDnsName,$SASURI)
   #get as object the uri and token for the blob SAS
    $url,$SAStoken,$SAStoken=$null
    $url=$($SASURI.value)
    $SAStoken=$SASURI.value.Split("?",2)[1]
    $sasobject=New-Object -TypeName psobject
    $sasobject | Add-Member -MemberType NoteProperty -Name URL -Value "$URL"
    $sasobject | Add-Member -MemberType NoteProperty -Name SASToken -Value "$SAStoken"
    $sasobject | Add-Member -MemberType NoteProperty -Name Machinename -Value
$package.Key
    $sasobjectjson=ConvertTo-Json $sasobject
    #call the http trigger in the logic app
    $LogicAppInfo = Invoke-WebRequest -Uri $LogicAppsUrl -Headers @{
    "Content-Type" = "application/json"
    } -Method Post -Body $sasobjectjson  -UseBasicParsing
    Start-Sleep -Seconds 15
    }
```

As can be seen in the last line of this code, it will call a logic app, that will download the zip containing the package, and extract into blob. Due to a limitation of the logic app code to extract zip files that has a maximum of 100 files in the zip, we will use this function that do not have this limit and was very easy to deploy on Azure. The delay in the logic apps allows for the function to kickin and extract the contents of the zip file.

GitHub - FBoucher/AzUnzipEverything: A simple Azure Function to Unzip files from a blob storage to another one



The code of the logic apps is as follows

```
{
    "definition": {
        "$schema": "https://schema.management.azure.com/providers/Microsoft.Logic
/schemas/2016-06-01/workflowdefinition.json#",
        "actions": {
            "Copy_blob_(V2)": {
                "inputs": {
                    "headers": {
                        "ReadFileMetadataFromServer": true
                    },
                    "host": {
                        "connection": {
                            "name": "@parameters('$connections')['azureblob_1']['
connectionId']"
                        }
                    },
                    "method": "post",
                    "path": "/v2/datasets/@{encodeURIComponent(encodeURIComponent
('AccountNameFromSettings'))}/copyFile",
                    "queries": {
                        "destination": "scheduledtasks/@{triggerBody()?['machinen
ame']}@{body('Create_blob_(V2)')?['LastModified']}.csv",
                        "overwrite": true,
                        "queryParametersSingleEncoded": true,
                        "source": "output-
files/Scheduled Tasks/ScheduledTasks.csv"
                    }
                },
                "runAfter": {
                    "Delay": [
```

```
                        "Succeeded"
                    ]
                },
                "type": "ApiConnection"
            },
            "Create_blob_(V2)": {
                "inputs": {
                    "body": "@body('HTTP')",
                    "headers": {
                        "ReadFileMetadataFromServer": true
                    },
                    "host": {
                        "connection": {
                            "name": "@parameters('$connections')['azureblob_1']['
connectionId']"
                        }
                    },
                    "method": "post",
                    "path": "/v2/datasets/@{encodeURIComponent(encodeURIComponent
('AccountNameFromSettings'))}/files",
                    "queries": {
                        "folderPath": "/input-files",
                        "name": "@{triggerBody()?['machinename']}.zip",
                        "queryParametersSingleEncoded": true
                    }
                },
                "runAfter": {
                    "HTTP": [
                        "Succeeded"
                    ]
                },
                "runtimeConfiguration": {
                    "contentTransfer": {
                        "transferMode": "Chunked"
                    }
                },
                "type": "ApiConnection"
            },
            "Delay": {
                "inputs": {
                    "interval": {
                        "count": 10,
                        "unit": "Second"
                    }
                },
```

```
            "runAfter": {
                "Create_blob_(V2)": [
                    "Succeeded"
                ]
            },
            "type": "Wait"
        },
        "HTTP": {
            "inputs": {
                "method": "GET",
                "uri": "@triggerBody()?['URL']"
            },
            "runAfter": {},
            "type": "Http"
        }
    },
    "contentVersion": "1.0.0.0",
    "outputs": {},
    "parameters": {
        "$connections": {
            "defaultValue": {},
            "type": "Object"
        }
    },
    "triggers": {
        "manual": {
            "inputs": {
                "schema": {
                    "properties": {
                        "SASToken": {
                            "type": "string"
                        },
                        "URL": {
                            "type": "string"
                        },
                        "machinename": {
                            "type": "string"
                        }
                    },
                    "type": "object"
                }
            },
            "kind": "Http",
            "type": "Request"
        }
```

```
        }
    },
    "parameters": {
        "$connections": {
            "value": {
                "azureblob_1": {
                    "connectionId": "/subscriptions/<subscription
id>/resourceGroups/<resource
group>/providers/Microsoft.Web/connections/azureblob-1",
                    "connectionName": "azureblob-1",
                    "id": "/subscriptions//<subscription
id>/providers/Microsoft.Web/locations/westeurope/managedApis/azureblob"
                }
            }
        }
    }
}
```

With the data already in blob where we can store it long term, we need it to flow to blob storage, so:

1.- We create a data explorer cluster following:

Quickstart: Create an Azure Data Explorer cluster and database | Microsoft Docs

2.- We create a database and add example data to it, this article closely related to our scenario (does not consider the automation to scale though) reflect this.

Using Microsoft Defender for Endpoint during investigation | Microsoft 365 Security (m365internals.com)

3.- We add a connection from blob onto the data Explorer cluster, this Will deploy an eventhub

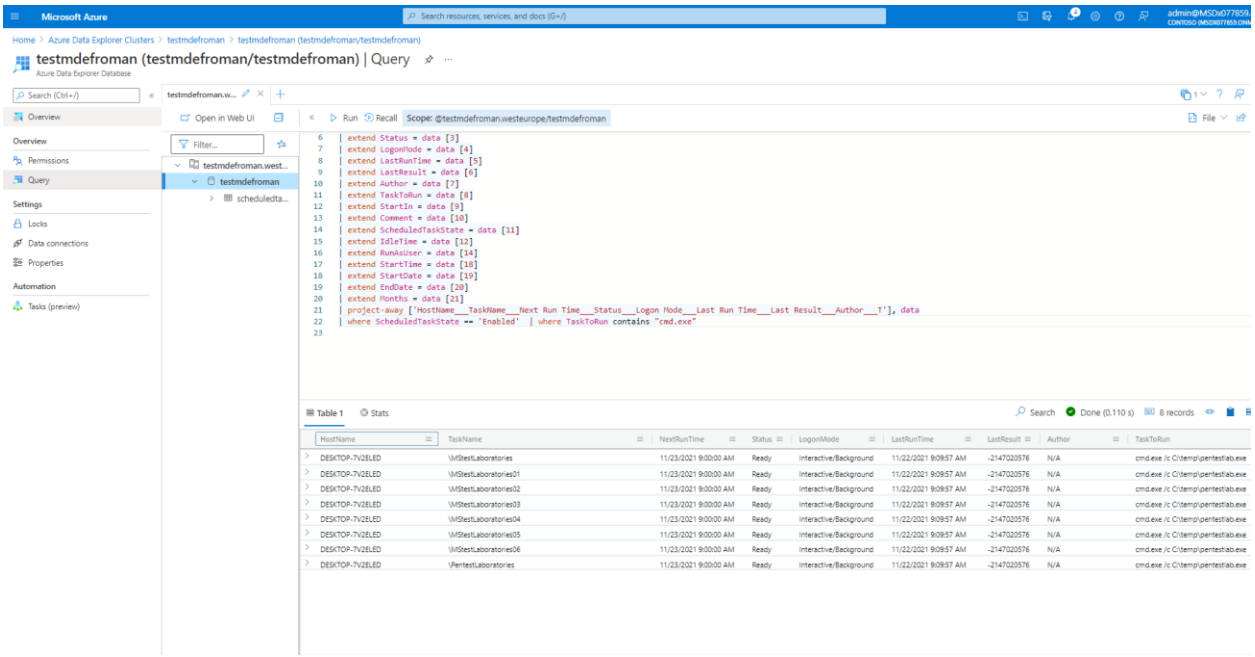
Ingest Azure Blobs into Azure Data Explorer | Microsoft Docs

4.- Data should be flowing into Data explorer, we should be able to query and parse as in this example.

```
scheduledtasks
| extend data = parse_csv(['HostName___TaskName___Next Run Time___Status___Logon
Mode___Last Run Time___Last Result___Author___T'])
| extend HostName = data [0]
| extend TaskName = data [1]
| extend NextRunTime = data [2]
| extend Status = data [3]
| extend LogonMode = data [4]
| extend LastRunTime = data [5]
| extend LastResult = data [6]
| extend Author = data [7]
```

```
| extend TaskToRun = data [8]
| extend StartIn = data [9]
| extend Comment = data [10]
| extend ScheduledTaskState = data [11]
| extend IdleTime = data [12]
| extend RunAsUser = data [14]
| extend StartTime = data [18]
| extend StartDate = data [19]
| extend EndDate = data [20]
| extend Months = data [21]
| project-
away ['HostName___TaskName___Next Run Time___Status___Logon Mode___Last Run Time_
__Last Result___Author___T'], data
| where ScheduledTaskState == 'Enabled'  | where TaskToRun contains "cmd.exe"
```

And that it is, ¡now we have the data parsed on our Data explorer ready to query and investigate!



That it is all for today, we hope you found it interesting, any doubt just let us know.