11 min read

Microsoft Sentinel

Deploy sentinel analytic rules with bicep and PowerShell

Robbe Van den Daele
Feb 9, 2023 • 11 min read

Photo by James Harrison / Unsplash

Introduction

The problem
- Manually enabling analytic rules takes a long time
- Not all of the out-of-the-box rules are good
- Keeping track of changes is a pain in the **s
- Many more

Subscribe

Introduction

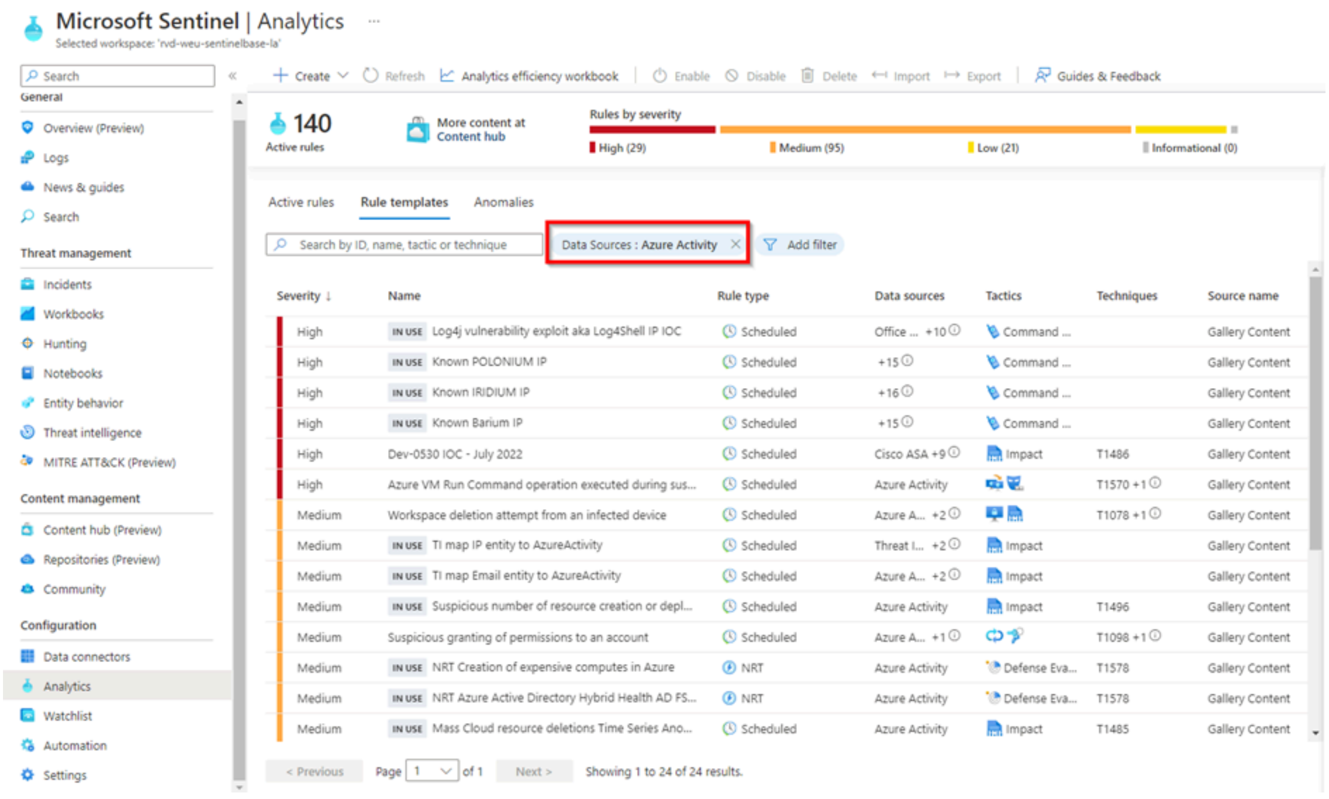
Managing a Microsoft Sentinel environment can get very complex, especially when you are responsible for managing multiple environments as MSSPs do. This is why Infrastructure as Code becomes very important. It makes sure that your environments are consistent, easy to roll out, and modular since you can standardize certain configurations over multiple customers.
In this post, I will be talking about how I managed to roll out Microsoft Sentinel analytic rules using Bicep and PowerShell. I used PowerShell so I was able to deploy analytic rules based on the data connectors that were enabled in the customer environments.

The problem

When you are an MSSP and want to manage multiple customers consistently, you will quickly find out that using IaC is the only option you have. A couple of issues I discovered when using the portal can be found in this section.

Manually enabling analytic rules takes a long time

For example, when you are onboarding a new customer and are connecting a data connector, you will want to enable the Out-Of-The-Box analytic rules regarding this data connector. To do this, you will have to go to analytic rules and filter on the data connector you just connected.



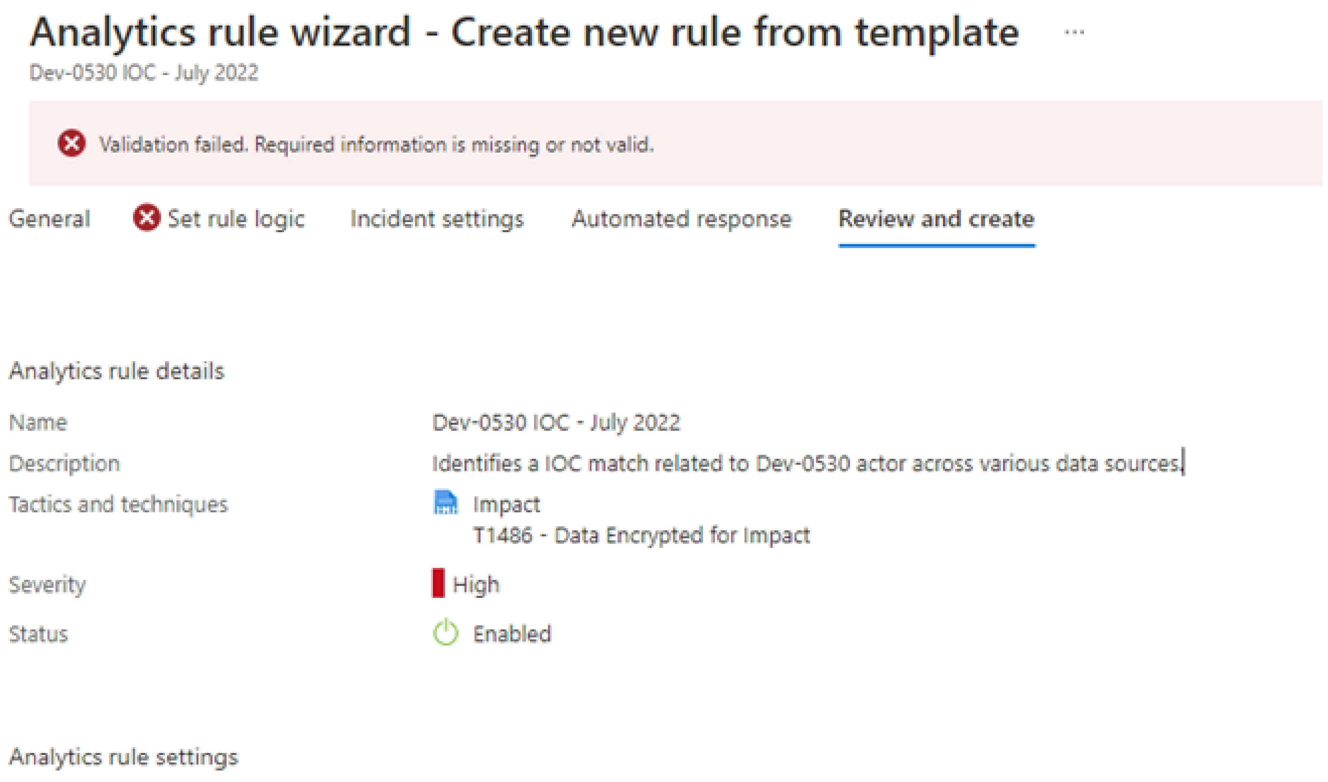
Severity ↓	Name	Rule type	Data sources	Tactics	Techniques	Source name
High	IN USE Log4j vulnerability exploit aka Log4Shell IP IOC	Scheduled	Office ... +10	Command ...		Gallery Content
High	IN USE Known POLONIUM IP	Scheduled	+15	Command ...		Gallery Content
High	IN USE Known IRIDIUM IP	Scheduled	+16	Command ...		Gallery Content
High	IN USE Known Barium IP	Scheduled	+15	Command ...		Gallery Content
High	Dev-0530 IOC - July 2022	Scheduled	Cisco ASA +9	Impact	T1486	Gallery Content
High	Azure VM Run Command operation executed during sus...	Scheduled	Azure Activity		T1570 +1	Gallery Content
Medium	Workspace deletion attempt from an infected device	Scheduled	Azure A... +2		T1078 +1	Gallery Content
Medium	IN USE TI map IP entity to AzureActivity	Scheduled	Threat I... +2	Impact		Gallery Content
Medium	IN USE TI map Email entity to AzureActivity	Scheduled	Azure A... +2	Impact		Gallery Content
Medium	IN USE Suspicious number of resource creation or depl...	Scheduled	Azure Activity	Impact	T1496	Gallery Content
Medium	Suspicious granting of permissions to an account	Scheduled	Azure A... +1		T1098 +1	Gallery Content
Medium	IN USE NRT Creation of expensive computes in Azure	NRT	Azure Activity	Defense Eva...	T1578	Gallery Content
Medium	IN USE NRT Azure Active Directory Hybrid Health AD FS...	NRT	Azure Activity	Defense Eva...	T1578	Gallery Content
Medium	IN USE Mass Cloud resource deletions Time Series Ano...	Scheduled	Azure Activity	Impact	T1485	Gallery Content

< Previous Page 1 of 1 Next > Showing 1 to 24 of 24 results.

Once you have done this, you need to click on each analytic rule, go to the 'review and create page', and create the analytic rule. If Azure has a good day, the portal refreshes after about three seconds and the rule is created. Know that Azure can sometimes be a lot slower, which makes the waiting process sometimes frustrating. After testing this process a couple of times, I found out that **enabling one analytic rule costs you four clicks and about 25 seconds**. This means that for a data connector like the Azure Activity where 24 Out Of The Box rules are available, you will have to **click 96 times** in the portal and need about **600 seconds (10 minutes).** If you have an environment where 200 analytic rules are enabled, this would take **800 clicks** and **5000 seconds (83 minutes).** If you would use IaC to deploy these analytic rules, all of the analytic rules would get rolled out in a matter of minutes. And even better, you can get a coffee while you are waiting on the deployment to finish.

Not all of the out-of-the-box rules are good

When you try to enable a couple of analytic rules, you will find out that not all analytic rules are working correctly. Some analytic rules have incorrect entity mappings, incorrect queries, or just don't fit your customer's needs. Below you find for example the Out Of The Box rule called 'Dev-0530 IOC – July 2022' which fails to validate for as long as I know the rule. In this case, it is because of the incorrect query mapping.



Analytics rule wizard - Create new rule from template

Dev-0530 IOC - July 2022

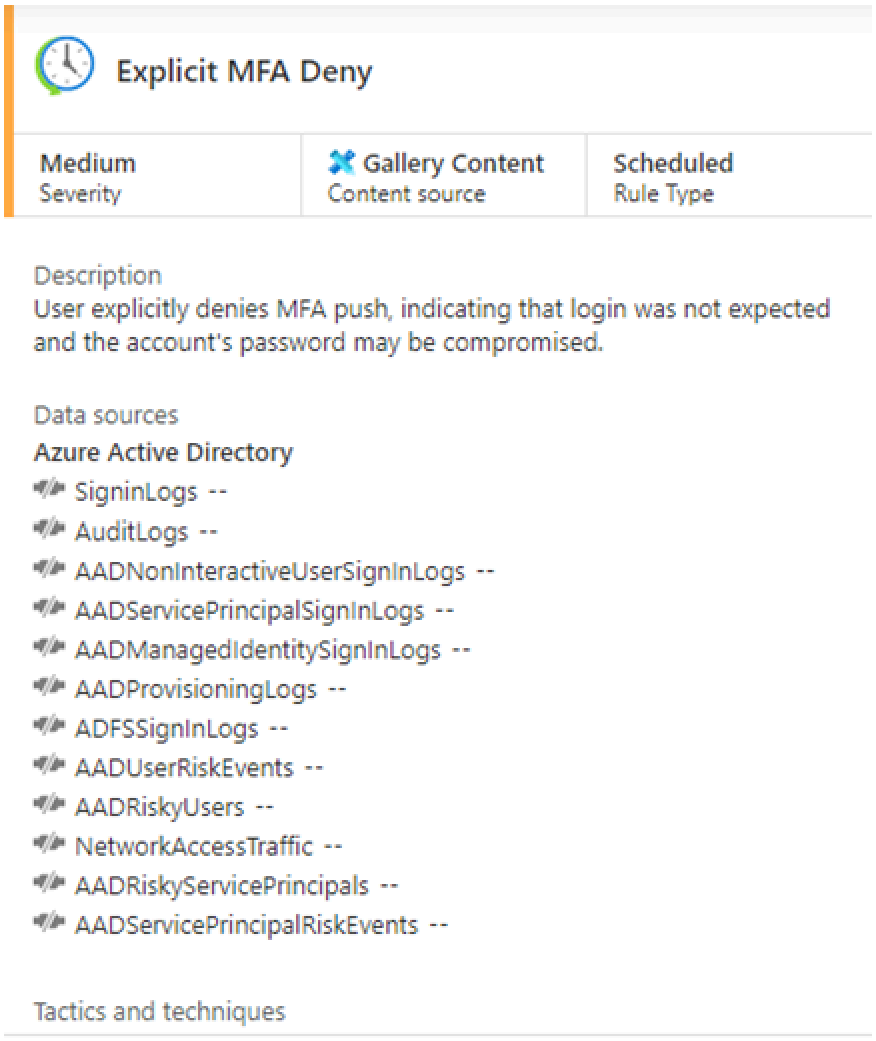
Validation failed. Required information is missing or not valid.

Analytics rule details

Name	Dev-0530 IOC - July 2022
Description	Identifies a IOC match related to Dev-0530 actor across various data sources.
Tactics and techniques	Impact T1486 - Data Encrypted for Impact
Severity	High
Status	Enabled

Analytics rule settings

Another example is the 'Explicit MFA Deny' rule. This rule creates an incident when a user clicks on the deny button in their authenticator app instead of clicking the approve button. In a lot of environments, you will find out that this will generate a lot of false positives. This is why you probably want to set a threshold, where an incident is only created when a user does this multiple times in a certain time frame.


Explicit MFA Deny

Medium	Gallery Content	Scheduled
Severity	Content source	Rule Type

Description
User explicitly denies MFA push, indicating that login was not expected and the account's password may be compromised.

Data sources
Azure Active Directory
SigninLogs --
AuditLogs --
AADNonInteractiveUserSignInLogs --
AADServicePrincipalSignInLogs --
AADManagedIdentitySignInLogs --
AADProvisioningLogs --
ADFSSignInLogs --
AADUserRiskEvents --
AADRiskyUsers --
NetworkAccessTraffic --
AADRiskyServicePrincipals --
AADServicePrincipalRiskEvents --

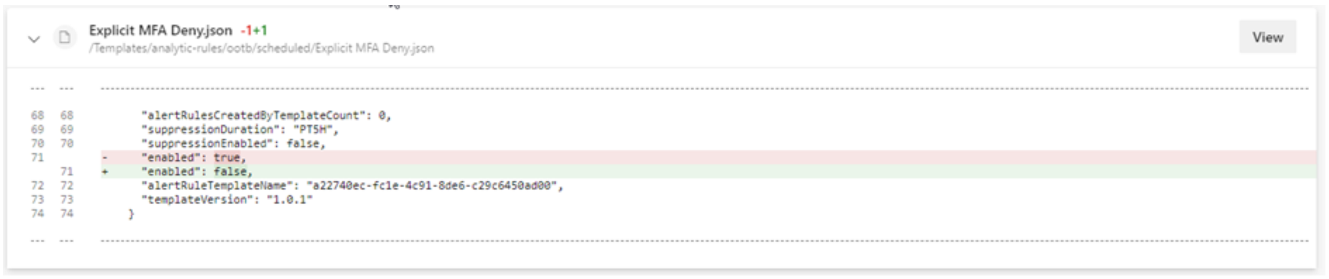
Tactics and techniques

You are probably wondering how IaC is going to fix this issue. If you make these changes all the time in multiple environments, it would be much more efficient if you make this change once, describe it as a template, and deploy this template for every customer that wants to use this rule. This way customers have all of the fine-tuning you already did as an MSSP at the very beginning of the Sentinel setup. If you fail to do this, you probably have a lot of false positives in a new deployment that you already encountered in other customer environments.

Keeping track of changes is a pain in the **s

When fine-tuning analytic rules in the portal, it is very hard to keep track of which analytic rules have custom fine-tuning, when you did it, and what changed compared to the older version. When you use IaC and couple a

GitHub or Azure DevOps repository to your project, you have much easier tracking of which rules are changed and what is changed. You are also able to organize your content like you want, making it easy to find analytic rules that are finetuned for one specific customer for example.


Explicit MFA Deny.json -1+1
/Templates/analytic-rules/ootb/scheduled/Explicit MFA Deny.json
View

```
        "alertRulesCreatedByTemplateCount": 0,
        "suppressionDuration": "PT5H",
        "suppressionEnabled": false,
-       "enabled": true,
+       "enabled": false,
        "alertRuleTemplateName": "a22740ec-fc1e-4c91-8de6-c29c6450ad00",
        "templateVersion": "1.0.1"
      }
```

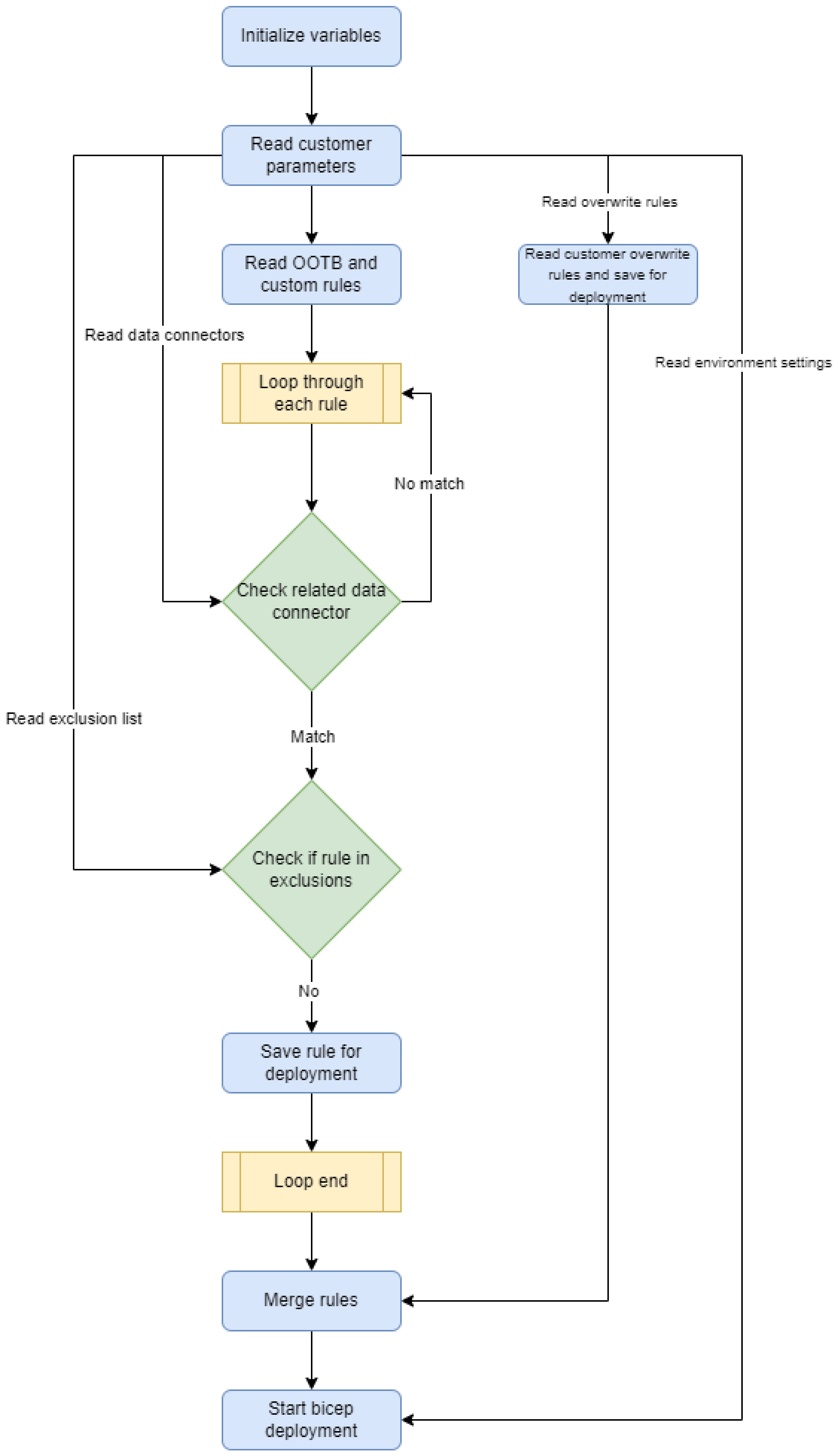
Many more

There are many more reasons why you would want to migrate your analytic rules to a DevOps environment:

- Creating approval flows using pipelines
- Do additional analytic rule checking before deployment
- Creating conditions when deploying analytic rules
- Using a custom auto-update script
- Auto-deploy all analytic rules based on data connector

How I did it

To be honest, I was not able to deploy sentinel analytic rules in the way I wanted just by using bicep. I created a PowerShell script that generates bicep parameter files based on the customer's needs and used those parameter files as input to my bicep scripts. The functional diagram of the code can be found below:


Initialize variables
Read customer parameters
Read OOTB and custom rules
Loop through each rule
Check related data connector
No match
Match
Check if rule in exclusions
No
Save rule for deployment
Loop end
Merge rules
Start bicep deployment
Read data connectors
Read exclusion list
Read overwrite rules
Read customer overwrite rules and save for deployment
Read environment settings

The PowerShell script makes sure that based on the data connector the customer has, all the related analytic rules are rolled out. In some cases, certain analytic rules don't need to be deployed although there is a related data connector, which is why we use an exclusion file. The overwrites file will contain analytic rules that have customer-specific finetuning.

Customer parameters

I defined a couple of customer-specific parameters that are needed for the deployment. These are:

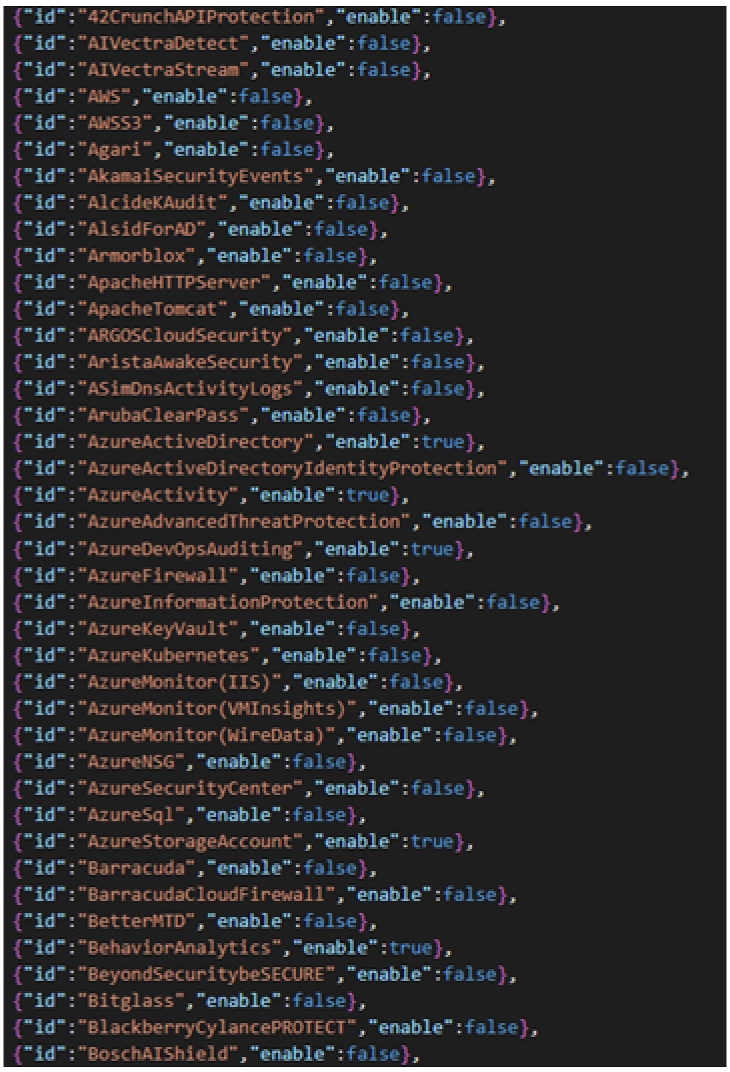
- resourcegroupName – Resource group of Sentinel workspace
- workspaceName – Sentinel workspace name
- overwritesPath – This is the path to the folder that contains all the overwrite rules of a customer.
- rulesParameterFile – The file that contains the data connectors the customer has
- ruleExclusionsFile – The file that contains rule IDs that do not need to be deployed
- location – The location of the bicep deployment

All these parameters are expected in the script to be able to start the deployment.

Rules Parameter File

The Rule Parameter File is a customer-specific file that contains all of the possible data connectors together with a property that shows if the data connector is enabled. It is a JSON Array with multiple JSON Objects, that I have built in the following schema:

```
[
    {“id”:”dataconnector1”,”enable”:true},
    {“id”:”dataconnector2”,”enable”:false},
    …
]
```

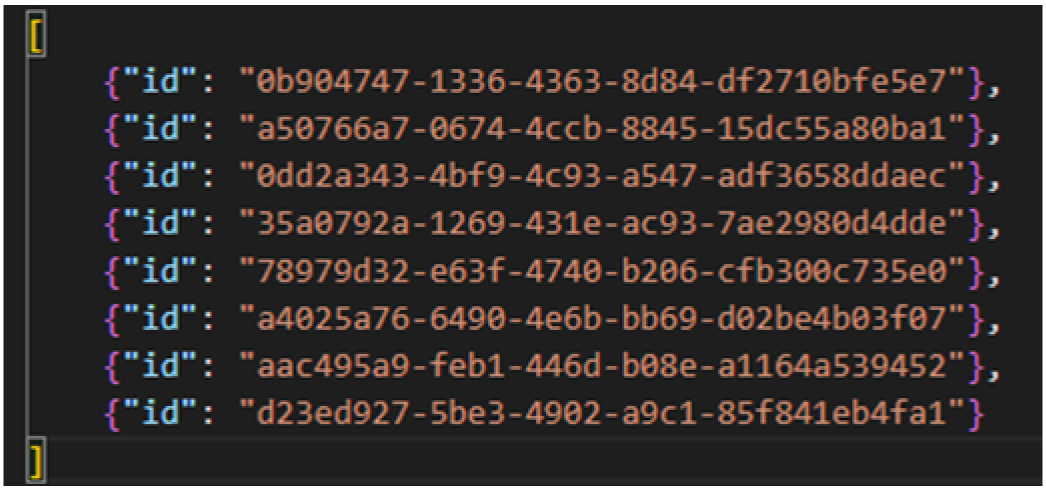


```
{"id":"42CrunchAPIProtection","enable":false},
{"id":"AIVectraDetect","enable":false},
{"id":"AIVectraStream","enable":false},
{"id":"AWS","enable":false},
{"id":"AWSS3","enable":false},
{"id":"Agari","enable":false},
{"id":"AkamaiSecurityEvents","enable":false},
{"id":"AlcideKAudit","enable":false},
{"id":"AlsidForAD","enable":false},
{"id":"Armorblox","enable":false},
{"id":"ApacheHTTPServer","enable":false},
{"id":"ApacheTomcat","enable":false},
{"id":"ARGOSCloudSecurity","enable":false},
{"id":"AristaAwakeSecurity","enable":false},
{"id":"ASimDnsActivityLogs","enable":false},
{"id":"ArubaClearPass","enable":false},
{"id":"AzureActiveDirectory","enable":true},
{"id":"AzureActiveDirectoryIdentityProtection","enable":false},
{"id":"AzureActivity","enable":true},
{"id":"AzureAdvancedThreatProtection","enable":false},
{"id":"AzureDevOpsAuditing","enable":true},
{"id":"AzureFirewall","enable":false},
{"id":"AzureInformationProtection","enable":false},
{"id":"AzureKeyVault","enable":false},
{"id":"AzureKubernetes","enable":false},
{"id":"AzureMonitor(IIS)","enable":false},
{"id":"AzureMonitor(VMInsights)","enable":false},
{"id":"AzureMonitor(WireData)","enable":false},
{"id":"AzureNSG","enable":false},
{"id":"AzureSecurityCenter","enable":false},
{"id":"AzureSql","enable":false},
{"id":"AzureStorageAccount","enable":true},
{"id":"Barracuda","enable":false},
{"id":"BarracudaCloudFirewall","enable":false},
{"id":"BetterMTD","enable":false},
{"id":"BehaviorAnalytics","enable":true},
{"id":"BeyondSecuritybeSECURE","enable":false},
{"id":"Bitglass","enable":false},
{"id":"BlackberryCylancePROTECT","enable":false},
{"id":"BoschAIShield","enable":false},
```

I based the creation of this file on the following GitHub page: https://github.com/Azure/Azure-Sentinel/blob/master/.script/tests/detectionTemplateSchemaValidation/ValidConnectorIds.json . Here you can find all of the valid connector ids that are in Sentinel.

Rule Exclusion File

The Rule Exclusion File is a file that contains the analytic rule names (in fact, these are GUID ids but more on that later) that should not be deployed, even though it contains the data connector ids that the customer uses in their environment.



```
[
    {"id": "0b904747-1336-4363-8d84-df2710bfe5e7"},
    {"id": "a50766a7-0674-4ccb-8845-15dc55a80ba1"},
    {"id": "0dd2a343-4bf9-4c93-a547-adf3658ddaec"},
    {"id": "35a0792a-1269-431e-ac93-7ae2980d4dde"},
    {"id": "78979d32-e63f-4740-b206-cfb300c735e0"},
    {"id": "a4025a76-6490-4e6b-bb69-d02be4b03f07"},
    {"id": "aac495a9-feb1-446d-b08e-a1164a539452"},
    {"id": "d23ed927-5be3-4902-a9c1-85f841eb4fa1"}
]
```

This was needed since we want to deploy analytic rules that are based on the Threat Intelligence data connector. There are a couple of rules related to this connector, that also query tables of data connectors a customer does not need. Since all analytic rules are tested and validated before deployment, analytic rules with non-existing tables forced the deployment to fail. To solve this, we add the analytic rule names that do not need to be deployed in this file.

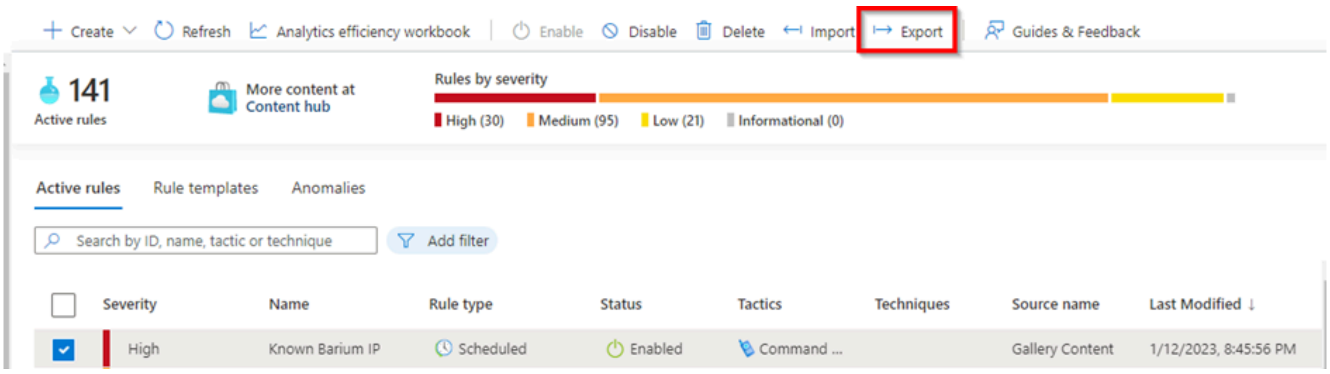
Overwrites Path

In the overwrites path we store all of the analytic rules that contain customer-specific finetuning. By default, our script will deploy all out-of-the-box analytic rules that are present. When an analytic rule needs specific finetuning, we place them in this overwrite path and deploy these rules after the OOTB deployments. **It is important to note that the name of the analytic rules need to be the same in the OOTB path and the Overwrites path for the rule to be overwritten**. More on how to do this is described in the following chapters.

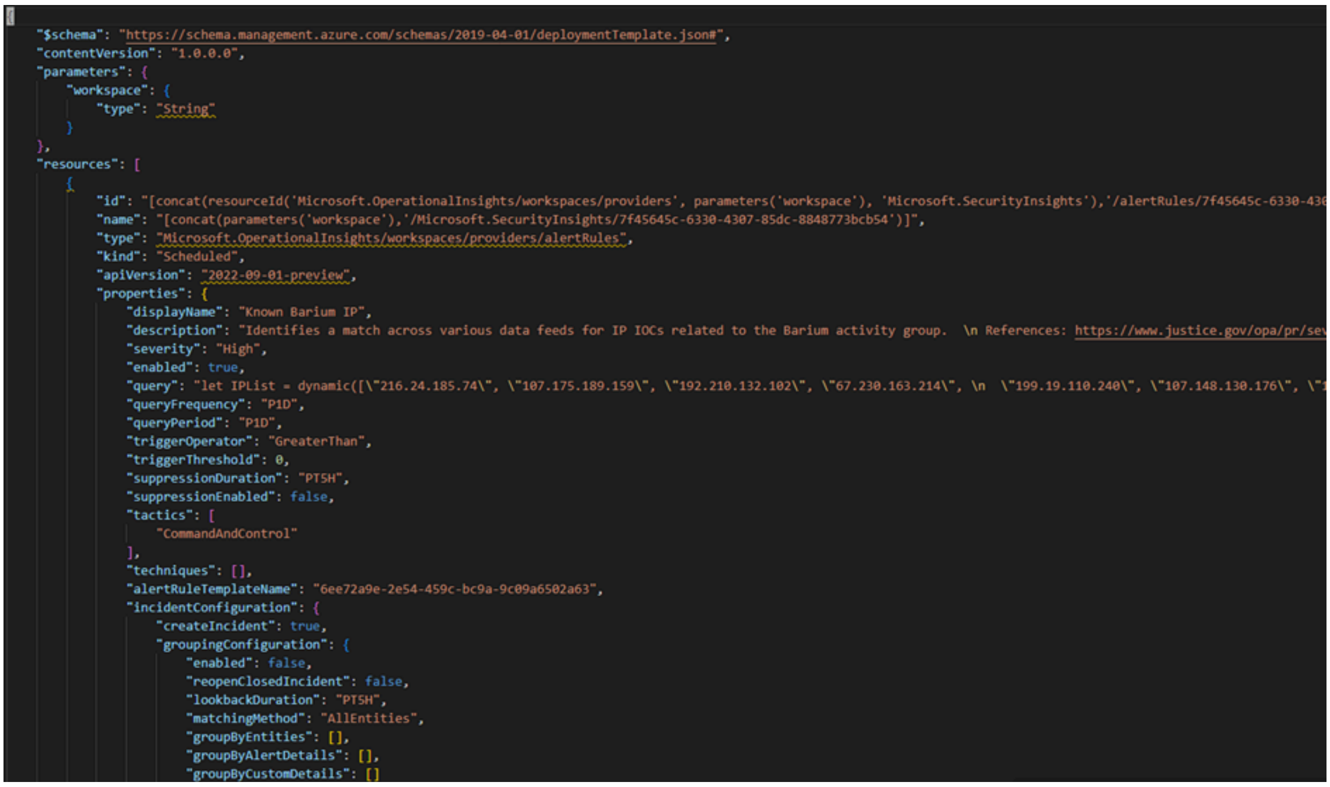
Analytic rule properties

We talked a couple of times about the path where OOTB rules and Overwrite rules are kept. But how are analytic rules described and what format are they in?

When you export an analytic rule via the Azure portal, you will find out that the analytic rule is described in JSON format. If you have some experience with ARM, you will even notice that it is in fact an ARM template!



	Severity	Name	Rule type	Status	Tactics	Techniques	Source name	Last Modified ↓
✓	High	Known Barium IP	Scheduled	Enabled	Command ...		Gallery Content	1/12/2023, 8:45:56 PM

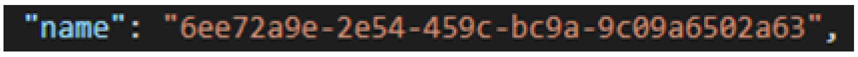


```
{
    "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "workspace": {
            "type": "String"
        }
    },
    "resources": [
        {
            "id": "[concat(resourceId('Microsoft.OperationalInsights/workspaces/providers', parameters('workspace'), 'Microsoft.SecurityInsights'),'/alertRules/7f45645c-6330-43
            "name": "[concat(parameters('workspace'),'/Microsoft.SecurityInsights/7f45645c-6330-4307-85dc-8848773bcb54')]",
            "type": "Microsoft.OperationalInsights/workspaces/providers/alertRules",
            "kind": "Scheduled",
            "apiVersion": "2022-09-01-preview",
            "properties": {
                "displayName": "Known Barium IP",
                "description": "Identifies a match across various data feeds for IP IOCs related to the Barium activity group.  \n References: https://www.justice.gov/opa/pr/sev
                "severity": "High",
                "enabled": true,
                "query": "let IPList = dynamic([\"216.24.185.74\", \"107.175.189.159\", \"192.210.132.102\", \"67.230.163.214\", \n  \"199.19.110.240\", \"107.148.130.176\", \"1
                "queryFrequency": "P1D",
                "queryPeriod": "P1D",
                "triggerOperator": "GreaterThan",
                "triggerThreshold": 0,
                "suppressionDuration": "PT5H",
                "suppressionEnabled": false,
                "tactics": [
                    "CommandAndControl"
                ],
                "techniques": [],
                "alertRuleTemplateName": "6ee72a9e-2e54-459c-bc9a-9c09a6502a63",
                "incidentConfiguration": {
                    "createIncident": true,
                    "groupingConfiguration": {
                        "enabled": false,
                        "reopenClosedIncident": false,
                        "lookbackDuration": "PT5H",
                        "matchingMethod": "AllEntities",
                        "groupByEntities": [],
                        "groupByAlertDetails": [],
                        "groupByCustomDetails": []
```

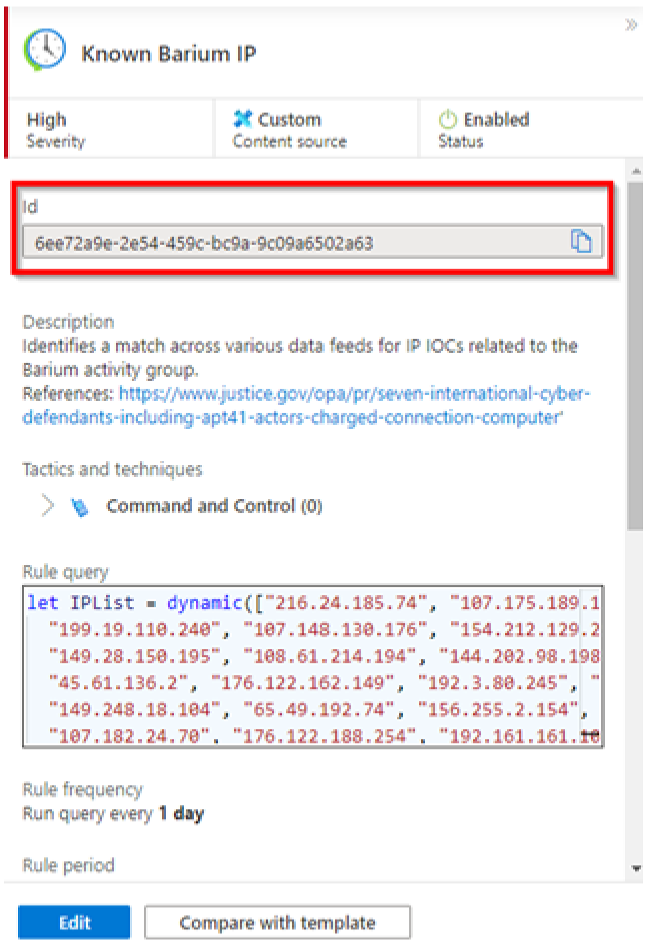
You can see in the template above that the ARM template has a property called “resources”. This is a JSON Array that contains all of the exported analytic rules you selected in the Azure Portal. Each JSON object in this array is the JSON representation of an analytic rule, which means you eventually only need these JSON Objects.

Now that you have a JSON export of the analytic rules, it is time to take a look at a couple of properties. There are a lot of properties used to describe an analytic rule. There in fact too much to talk about. If you want to learn more about all of the properties, you can visit the SecurityInsights alert rules documentation. However, below you will find an explanation of the most important analytic rule properties.

- Id – The ID field is not really used during a bicep deployment of the analytic rule. It just forms the unique ID together with your subscription and workspace when you export the rule to an ARM template. **You can ignore this property**.
- Name – The name field is a GUID and is the name of your analytic rule. It is unique in a Sentinel workspace. When you deploy an analytic rule to your environment where there is already a rule with the same name, the analytic rule will be overwritten. You can also find the name of the analytic rule in the ‘id’ field in the Azure Portal (I know, pretty complex right)



```
"name": "6ee72a9e-2e54-459c-bc9a-9c09a6502a63",
```


Known Barium IP

High
Severity

Custom
Content source

Enabled
Status

Id
6ee72a9e-2e54-459c-bc9a-9c09a6502a63

Description
Identifies a match across various data feeds for IP IOCs related to the Barium activity group.
References: https://www.justice.gov/opa/pr/seven-international-cyber-defendants-including-apt41-actors-charged-connection-computer'

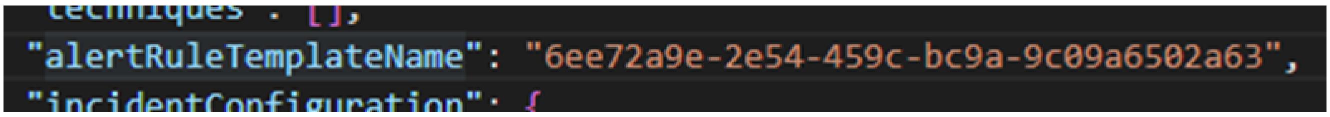
Tactics and techniques
Command and Control (0)

Rule query
```
let IPList = dynamic(["216.24.185.74", "107.175.189.1
  "199.19.110.240", "107.148.130.176", "154.212.129.2
  "149.28.150.195", "108.61.214.194", "144.202.98.198
  "45.61.136.2", "176.122.162.149", "192.3.80.245", "
  "149.248.18.104", "65.49.192.74", "156.255.2.154",
  "107.182.24.70", "176.122.188.254", "192.161.161.10
```

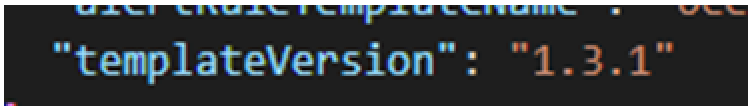
Rule frequency
Run query every **1 day**

Rule period

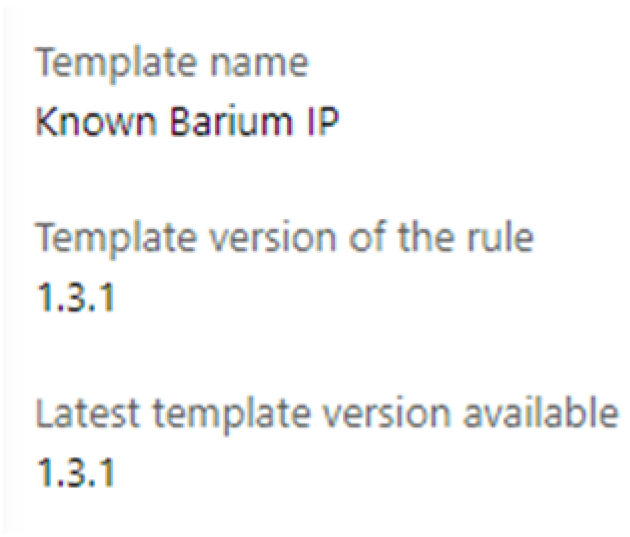
- alertRuleTemplateName – This property contains the name of the template on which the analytic rule is based. This is not a required property, since you can always create custom analytic rules from scratch. This field will be important if you want to track analytic rule versions with the latest template version.


```
"techniques": [],
"alertRuleTemplateName": "6ee72a9e-2e54-459c-bc9a-9c09a6502a63",
"incidentConfiguration": {
```

- templateVersion – The template version field contains the version of the template on which the analytic rule is based. When this version is lower than the latest version existing for the template, the analytic rule will get the ‘update available’ tag.



```
"templateVersion": "1.3.1"
```

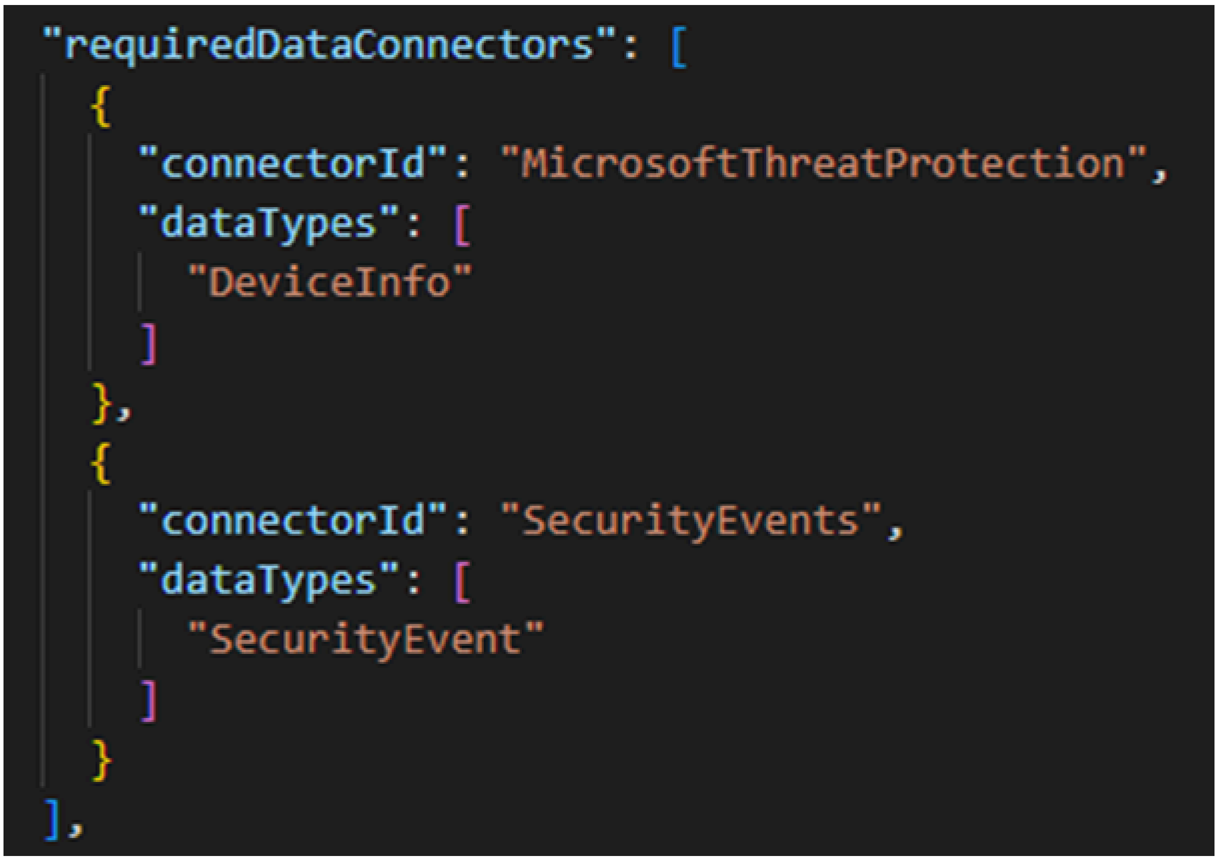


Template name
Known Barium IP

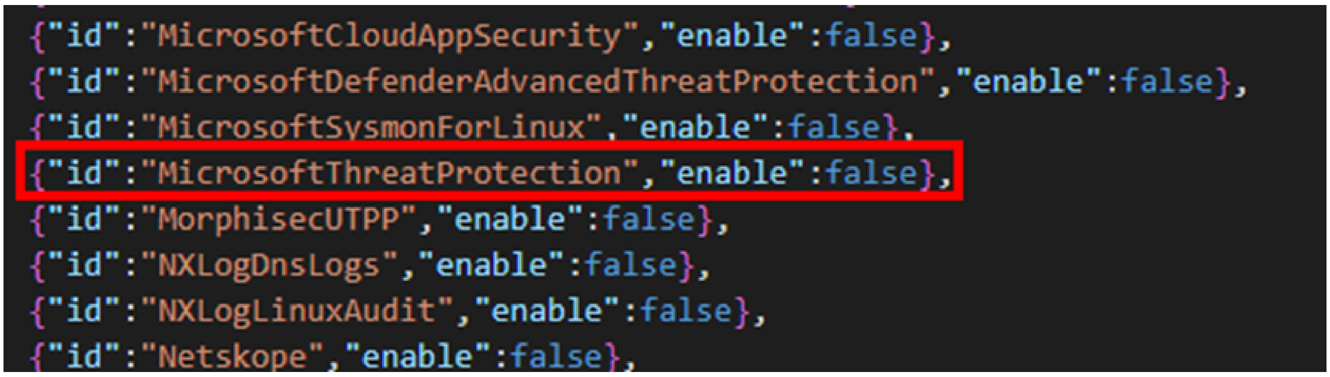
Template version of the rule
1.3.1

Latest template version available
1.3.1

- requiredDataConnectors – This property is an array that contains JSON objects regarding the data connectors the analytic rule uses. These objects contains the connectorId field, and the dataTypes field. The connectorId field is the field that we check to see if an analytic rule needs to be deployed for a customer, which means that these are the fields that are present in our Rules Parameter File!



```
"requiredDataConnectors": [
  {
    "connectorId": "MicrosoftThreatProtection",
    "dataTypes": [
      "DeviceInfo"
    ]
  },
  {
    "connectorId": "SecurityEvents",
    "dataTypes": [
      "SecurityEvent"
    ]
  }
],
```



```
{"id":"MicrosoftCloudAppSecurity","enable":false},
{"id":"MicrosoftDefenderAdvancedThreatProtection","enable":false},
{"id":"MicrosoftSysmonForLinux","enable":false},
{"id":"MicrosoftThreatProtection","enable":false},
{"id":"MorphisecUTPP","enable":false},
{"id":"NXLogDnsLogs","enable":false},
{"id":"NXLogLinuxAudit","enable":false},
{"id":"Netskope","enable":false},
```

All the other properties that can be found in the analytic rule template are the same properties you can find in the portal. Again, if you want more information about the other properties, I recommend checking the SecurityInsights alert rules documentation.

Bicep code

Assuming you already know something about bicep, you can probably imagine that we need two bicep files to deploy analytic rules. One bicep file is scoped at subscription level, the other file is scoped at the resource group that contains the Sentinel log analytics workspace. I named the first bicep file **main.bicep**, and the second bicep file **deploy-rules.bicep**.

Az-Context

Before you can start deploying bicep, you first need the set your environment for the deployment. You can use the Connect-AzAccount PowerShell command to log in to your tenant and choose the subscription where your sentinel environment resides.

main.bicep

The main.bicep file is the file that gets started by the deployment. In this file we do a couple of steps, which are the following:

1. Get the required parameters
   a. Resource group name
   b. Sentinel workspace name
2. Read the analytic rules JSON files via the loadJsonContent function (these are the files with the analytic rule properties in JSON that you should generate via powershell)
3. Set the target scope to subscription level
4. Get the existing resource group of the sentinel workspace
5. Deploy the OOTB analytic rules (a reference to deploy-rules.bicep)
6. Deploy the Overwrite analytic rules (a reference to deploy-rules.bicep)

```
        //--------------------
// Parameters
//--------------------
@description('Resource group name where the Sentinel workspace resides')
param resourceGroupName string

@description('Log analytics worksapce of Sentinel')
```

```
param workspaceName string


//--------------------
// Variables
//--------------------
var ootbRules = loadJsonContent('deployments/analytic-rules.json').resources
var overwriteRules = loadJsonContent('deployments/overwrite-rules.json').resources

//--------------------
// TragetScope
//--------------------
targetScope = 'subscription'


//--------------------
// Code
//--------------------
// Get existing resource group
resource sentinelResourceGroup 'Microsoft.Resources/resourceGroups@2021-04-01' exis
  name: resourceGroupName
}

// Deploy analytic rules
module deployOOTBRules 'deploy-rules.bicep' = {
  scope: sentinelResourceGroup
  name: 'deployOOTBRules'
  params: {
    workspaceName: workspaceName
    rules: ootbRules
  }
}

// Deploy overwrite rules
module deployOverwriteRules 'deploy-rules.bicep' = {
  scope: sentinelResourceGroup
  name: 'deployOverwriteRules'
  dependsOn: [ deployOOTBRules ]
  params: {
    workspaceName: workspaceName
    rules: overwriteRules
  }
}
```

Show on Github

The command that is used to start this file is

```
New-AzSubscriptionDeployment -TemplateFile "main.bicep" -TemplateParameterFile ".\de
```

This command will start the main.bicep file, with a file params.json as parameter file. The parameter file contains the parameters that we set in step 1, which are the resource group and sentinel workspace name. If you build a PowerShell script like I did to start the bicep deployment, this file should be generated in your PowerShell script (you can set the sentinel workspace and resource group name as required parameters in PowerShell).

```
$bicepParameterFile = @"
    {
        "`$schema": "https://schema.management.azure.com/schemas/2015-01-01/deployme
        "contentVersion": "1.0.0.0",
        "parameters": {
            "resourceGroupName": {
                "value": "$resourcegroupName"
            },
            "workspaceName": {
                "value": "$workspaceName"
            }
        }
    }
"@
$bicepParameterFile | Out-File ".\deployments\params.json"
```

deploy-rules.bicep

The deploy-rules.bicep file is eventually the file where analytic rules get deployed. Below you find the steps of the bicep file:

1. We first read the parameters for the deployment
   a. Sentinel workspace name
   b. Analytic rules in array format

2. Get the existing sentinel workspace by providing the workspace name
3. Deploying the analytic rules by using a for-loop. Here we can get the required properties for every rule which are
   a. Name
   b. Kind
   c. Properties (the object that contains the analytic rule properties)

```
        //--------------------
// Parameters
//--------------------
param workspaceName string
param rules array

//--------------------
// Code
//--------------------
// Get the existing la
resource sentinelWorkspace 'Microsoft.OperationalInsights/workspaces@2022-10-01' ex
  name: workspaceName
}

// Deploy rules
resource analyticRuleDeployment 'Microsoft.SecurityInsights/alertRules@2022-10-01-pr
  name: rule.name
  scope: sentinelWorkspace
  kind: rule.kind
  properties: rule.properties
}]
```

Show on Github

Get current out of the box analytic rules

Below you will find a script that gets all of the out-of-the-box analytic rules. The script expects the resource group name of your sentinel workspace and the name of the log analytics workspace. It then created a bearer token based on your az-context parameters (so make sure your context is correct). In the next step, the script gets all the template rules by using the Azure Management API and adds the required properties that are needed for a bicep deployment for each analytic rule (the properties of a template analytic rule are not the same as the properties you find in an exported existing analytic rule). Finally, the script saves the analytic rules in JSON format in different folders based on the alert rule types.

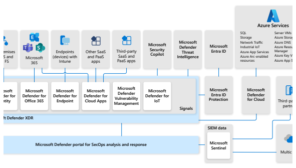
```
        [CmdletBinding()]
param (
    [Parameter (Mandatory=$true)]
    [string] $resourceGroupName,
    [Parameter (Mandatory=$true)]
    [string] $workspaceName
)


# --------------------
# Set parameters
# --------------------
# Get the context
$context = Get-AzContext
$azureProfile = [Microsoft.Azure.Commands.Common.Authentication.Abstractions.AzureRm
$profileClient = New-Object -TypeName Microsoft.Azure.Commands.ResourceManager.Comm
# Save auth header and subscription
$token = $profileClient.AcquireAccessToken($context.Subscription.TenantId)
$authHeader = @{
    'Content-Type'  = 'application/json'
    'Authorization' = 'Bearer ' + $token.AccessToken
}
$SubscriptionId = $context.Subscription.Id
# Create body
$body = @{
    "subscriptions" = @($SubscriptionId)
}
# Generate URL
$uri = "https://management.azure.com/subscriptions/$SubscriptionId/resourceGroups/$r


# --------------------
# Save templates
# --------------------
# Get templates and convert them
```

```
$verdict = Invoke-RestMethod -Uri $uri -Method Get -Headers $authHeader -Body $body
$templates = $verdict.value | ConvertTo-Json -Depth 10 | ConvertFrom-Json
# Loop through each template
$templates | ForEach-Object {
    # Replace illegal characters
    $displayName = $_.properties.displayName
    if ($displayName -match '/') {
        $displayname = $displayName.replace('/', ' or ') | Out-Null
    }
    # Add required parameter that is not included in template
    $_.properties | Add-Member -NotePropertyName "suppressionDuration" -NoteProperty
    $_.properties | Add-Member -NotePropertyName "suppressionEnabled" -NoteProperty
    $_.properties | Add-Member -NotePropertyName "enabled" -NotePropertyValue $true
    $_.properties | Add-Member -NotePropertyName "alertRuleTemplateName" -NotePrope
    $_.properties | Add-Member -NotePropertyName "templateVersion" -NotePropertyVal
    # Save correct kind under correct folder
    if ($_.kind -eq "MLBehaviorAnalytics") { $_ | ConvertTo-Json -Depth 10 | Out-Fi
    elseif ($_.kind -eq "Scheduled") { $_ | ConvertTo-Json -Depth 10 | Out-File -Fo
    elseif ($_.kind -eq "ThreatIntelligence") { $_ | ConvertTo-Json -Depth 10 | Out
    elseif ($_.kind -eq "MicrosoftSecurityIncidentCreation") { $_ | ConvertTo-Json
    elseif ($_.kind -eq "Fusion") { $_ | ConvertTo-Json -Depth 10 | Out-File -Force
    elseif ($_.kind -eq "NRT") { $_ | ConvertTo-Json -Depth 10 | Out-File -Force -F
    else { $_ | ConvertTo-Json -Depth 10 | Out-File -Force -FilePath "$($displayName
}
```

Show on Github



Transition from Microsoft

Detecting non-privileged

MDE Device Discovery -

Sentinel to Defender XDR - Practical challenges

Introduction Microsoft announced on the...

Jul 4, 2025 12 min read

Windows Hello abuse

Introduction I recently followed a live session of Dirk...

Apr 26, 2025 16 min read

Improving the monitored network page

Introduction This blogpost is probably the first ...

Mar 19, 2025 6 min read