Release 4.1.0 Artifacts - Power BI Connector with CM & VRM

Bitsight

Exported on 2025-04-16 11:42:04

Table of Contents

1 Project scope 5

1.1 APIs configured 5

1.2 Initial requirements 6

1.3 Change log 6

2 General Technical Overview - Release 4.1.0 7

2.1 Introduction 7

2.2 Authentication 7

2.2.1 Headers 7

2.3 Folder Structure 8

2.4 Entities 10

2.5 Schemas 12

2.6 Get Entity and Apply Schema 12

2.6.1 Purpose 15

2.6.2 Function Logic 15

2.7 Diagram Flow of Data 15

3 PowerBI Connector for Bitsight - Release 4.1.0 17

3.1 17

3.2 Overview 17

3.3 Technologies Used 17

3.4 Bitsight APIs 17

3.5 Architecture Diagrams 18

3.6 Certification 19

4 Manual Installation of a Power BI Connector - Release 4.1.0 20

4.1 Overview 20

4.1.1 Why Manual Installation is Needed? 20

4.2 Prerequisites 21

4.2.1 Step-by-Step Installation 21

4.3 Best Practices 22

5 Bitsight Continuous Monitoring API - Release 4.1.0 23

5.1 Introduction 23

5.2 Helpful Variables 23

5.3 Endpoints for Continuous Monitoring API 24

5.4 Pagination 27

5.4.1 What is Pagination? 27

5.4.2 Explanation of the Code 28

5.4.3 Summary 30

6 Bitsight Vendor Ratings Management API - Release 4.1.0 32

6.1 Introduction 32

6.2 Power BI Data Modelling 33

6.3 Entities 33

6.3.1 Endpoints for Vendor Rating Management API 33

6.4 Code explanation 37

6.4.1 All Vendors 37

6.4.2 Vendor Details 37

6.4.3 Vendor Rating 39

6.4.4 Vendor Info 41

6.4.5 Vendor Contacts 42

6.4.6 Vendor Internal Contacts 43

6.4.7 Vendor Findings 44

6.4.8 Findings Details 45

6.4.9 Internal Questionnaires: 46

6.4.10 Internal Questionnaires Categories 48

6.4.11 Internal Questionnaires Questions 50

6.4.12 Internal Questionnaires Answers 54

6.4.13 Internal Questionnaires Progress 56

6.4.14 External Questionnaires 59

6.4.15 External Questionnaires Categories 59

6.4.16 External Questionnaires Answers 59

6.4.17 External Questionnaires Questions 59

6.4.18 External Questionnaires Progress 59

6.4.19 Vendor Requirements 59

6.4.20 Vendor Requirements Details 61

6.4.21 Vendor Lifecyle 65

6.5 Performance Issues 66

7 Xpand IT Team 67

|  |  |
| --- | --- |
| Content | File / Link |
| Binaries | [Connector.mez (v4.1.0)](https://apps.xpand-it.com/confluence/download/attachments/463176526/Connector.mez?api=v2&modificationDate=1744726365416&version=1) |
| Manuals | [Manual Installation of a Power BI Connector](https://apps.xpand-it.com/confluence/display/Bitsight/Manual%2BInstallation%2Bof%2Ba%2BPower%2BBI%2BConnector) |

# Project scope

Adapt the existing Bitsight Power BI Custom Connector, developed by the Xpand IT in the past, using Power Query, to include some of the VRM product data that will be obtained by Bitsight APIs. This connector will enable BitSight’s customers to easily explore and share BitSight’s platform information in Power BI.

|  |  |
| --- | --- |
| **Stage** | **Description** |
| Analysis & Setup | Initial analysis to define requirements, setup connections and support new data source setup for 24 endpoints of VRM |
| API Authentication | Qualify the BitSight’s API authentication using an API token. Validate access to VRM’s API endpoints list. |
| Connector Navigation | Develop a connector navigation screen that will allow the user to choose the tables to import in Power BI Desktop with user friendly table names and data previews using default features. |
| VRM API endpoints | Develop access to 24 new API endpoints calls and its JSON response processing using Power Query. The new endpoint list provided via email corresponds to VRM product data. A few endpoints have dependencies on other endpoints. |
| Performance revision | Once the initial development is completed it will be done a performance revision over the solution to ensure the best UX over this aspect |
| Testing and Production support | Delivery of the solution & 2 iterations of testing / revisions with BitSight before official publication |
| Documentation | Documentation of the full solution developed to ensure future full ownership by Bitsight. |

* Connector: development for the Power BI Connector will be done using Power Query / M language and only with Power BI default available features.
* UI/UX: will be developed using Power BI default available features.
* APIs:the access of APIs and the data obtained is part of BitSight’s ownership and responsibility. Pagination for the VRM’s API list provided is out of scope.
* Environment: only one solution environment is being considered
* Testing: including testing and UAT
* Re-utilization: we are assuming we will be re-using part of the existing Power BI Connector features
* Validation and Publication: Bitsight is responsible for testing the developments and aligning with Microsoft the requirements and the phase of publication of the new version of the connector. The Service provider will support these activities and implement the necessary changes within the scope.

## APIs configured

* [Endpoints for Vendor Rating Management API](https://apps.xpand-it.com/confluence/display/Bitsight/Bitsight%2BVendor%2BRatings%2BManagement%2BAPI%2B-%2BRelease%2B2.0#BitsightVendorRatingsManagementAPIRelease2.0-EndpointsforVendorRatingManagementAPI)
* [Endpoints for Continuous Monitoring API](https://apps.xpand-it.com/confluence/display/Bitsight/Bitsight%2BContinuous%2BMonitoring%2BAPI%2B-%2BRelease%2B2.0#BitsightContinuousMonitoringAPIRelease2.0-EndpointsforContinuousMonitoringAPI)

## Initial requirements

Can be found in this document shared by Bitsight - [PowerBI\_Tableau for Bitsight VRM.pptx](https://apps.xpand-it.com/confluence/download/attachments/463176526/PowerBI_Tableau%20for%20Bitsight%20VRM.pptx?api=v2&modificationDate=1744726365442&version=1)

## Change log

**Release Highlights (v4.1.0):**

* **📁 Folder Structure Refactor:**
Adjusted the folder structure to ensure backward compatibility with dashboards built using version **v3.9.0**. This prevents data source path mismatches for users upgrading directly from older versions that was happening in **v4.0.0**. See more in [Folder Structure](https://apps.xpand-it.com/confluence/display/Bitsight/General%2BTechnical%2BOverview%2B-%2BRelease%2B4.1#GeneralTechnicalOverviewRelease4.1-FolderStructure) .
* **🐛 Requirements Progress Table Pagination Fix:**
Resolved an issue causing errors when loading the **requirements progress** table.
Previously, the API call failed due to exceeding the URL limit when requesting data for too many vendors at once.
We've now implemented **forced pagination (in batches of 50 vendor IDs)** to ensure stable and reliable data retrieval.

# General Technical Overview - Release 4.1.0

* [Introduction](#scroll-bookmark-9)
* [Authentication](#scroll-bookmark-10)
	+ [Headers](#scroll-bookmark-11)
* [Folder Structure](#scroll-bookmark-12)
* [Entities](#scroll-bookmark-13)
* [Schemas](#scroll-bookmark-14)
* [Get Entity and Apply Schema](#scroll-bookmark-15)
	+ [Purpose](#scroll-bookmark-16)
	+ [Function Logic](#scroll-bookmark-17)
* [Diagram Flow of Data](#scroll-bookmark-18)

## Introduction

Quick introduction with an overall technical explanation on how the model works. For more information check functional detailed documentation.

## Authentication

Authentication for the Bitsight API is token-based; login credentials for the Bitsight VRM Platform will not work.

* All API requests must be made over HTTPS. Calls made over HTTP will fail.
* All requests must be authenticated.

How to get a Token

A user can create their own Token by going to <https://service.bitsighttech.com/> (https://service.bitsighttech.com/) Settings->Account scrolling down to the API Token sections. There you can generate a new token. If a token is compromised, a new one can always be generated, which will invalidate the previous one.

### Headers

In the integrations and data requests from Bitsight we use three API headers formats.

If there are any new integrations or changes, please include always:

|  |  |
| --- | --- |
| Name | Details |
| X-Bitsight-CONNECTOR-NAME-VERSION | Specify the name of the connector |
| X-Bitsight-CALLING-PLATFORM-VERSION | Identifies platform |
| X-BITSIGHT-CUSTOMER |  identify the mutual customer on whose behalf they are calling the API |

We have three different headers. The first one is used at the start of the application to get the customer’s name and to identify on whose behalf the API is being called.

|  |
| --- |
| FirstRequestHeaders = [ Authorization = "Basic " & apiKey, Accept = "application/json", #"Content-Type" = "application/json", #"X-BITSIGHT-CONNECTOR-NAME-VERSION" = "Security Ratings Connector v4.1.0", #"X-BITSIGHT-CALLING-PLATFORM-VERSION" = "PowerBI", #"X-BITSIGHT-CUSTOMER" = "GET Customer Name"]; |

The company name will be retrieved from the following function. After this, we can pass the company name in the headers to pull data from the APIs.

|  |
| --- |
| company\_name = GetMyCompanyName(); guidUrl = "https://api.bitsighttech.com/v1/users/current"; GetMyCompanyName = () as text => let source = Json.Document(Web.Contents(guidUrl, [ Headers = FirstRequestHeaders ])), name = source[customer][name], output = name in output; |

Since we have two integrations pulling data from two different Bitsight APIs, we have two different headers.

For the first implementation to get data from the **Bitsight API**:

|  |
| --- |
| DefaultRequestHeaders = [ Authorization = "Basic " & apiKey, Accept = "application/json", #"Content-Type" = "application/json", #"X-BITSIGHT-CONNECTOR-NAME-VERSION" = "Security Ratings Connector v4.1.0", #"X-BITSIGHT-CALLING-PLATFORM-VERSION" = "PowerBI", #"X-BITSIGHT-CUSTOMER" = company\_name]; |

And for the second implementation, to get data from the **VRM API**:

|  |
| --- |
| DefaultVRMRequestHeaders = [ Authorization = "Basic " & apiKey, Accept = "application/json", #"Content-Type" = "application/json", #"X-BITSIGHT-CONNECTOR-NAME-VERSION" = "VRM Connector v4.1.0", #"X-BITSIGHT-CALLING-PLATFORM-VERSION" = "PowerBI", #"X-BITSIGHT-CUSTOMER" = company\_name]; |

Note how the company\_name retrieved earlier is now sent in the headers.

## Folder Structure

The folder structure organizes the data and endpoints into logical groups, making it easy for users to navigate and access data from the Bitsight APIs. The structure assures that users in v3.9 can still refresh their data in the previously used tables without breaking the datasource path:

Folders in "Bitsight Security Ratings"

|  |  |  |
| --- | --- | --- |
| Folder Name | Key | Description |
| My Company | My Company | Data related to the user's company, such as findings and reports.     |
| Others | Others | Information about other companies relevant to the user. |
| My Infrastructure | My Infrastructure | Infrastructure-related data, such as IP and domain reports. |
| Portfolio | Portfolio | Portfolio metrics, including risk vectors and ratings. |
| Bitsight Vendor Risk Management API | Bitsight Vendor Risk Management API | Contains folders for managing vendor-related data, including portfolios, findings, and questionnaires. |

Subfolders in "Bitsight Vendor Risk Management API"

|  |  |  |
| --- | --- | --- |
| Folder Name | Key | Description |
| Contacts | Contacts | Information about internal and external vendor contacts. |
| Findings | Findings | Details of security findings associated with vendors. |
| Internal Questionnaires | Internal Questionnaires | Data on internal assessments conducted for vendors. |
| Lifecycle | Lifecycle | Information on the different lifecycle stages of a vendor. |
| Questionnaires | Questionnaires | Data on external questionnaires used for vendor evaluation. |
| Requirements | Requirements | Vendor-specific compliance and security requirements. |
| Vendors | Vendors | General information and details about vendors. |

Here we have an image of the navigation structure when the Bitsight PBI connector is selected:



## Entities

There is a root entities list declared in the code for each table that the user will be able to choose from the connector from the Navigation Table.

|  |
| --- |
| // Entities List for the Navigation TableRootEntities = #table({"entity\_order","entity\_name","endpoint","flag","url"}, { {0,"Details","company","d",companyURL}, {1,"Ratings History","company","h",companyURL}, {2,"Current Risk Vector Grades","company","g",companyURL}, {3,"Findings","findings","a",findingsURL}, {4,"Risk Vector History","rvhistory","a",rvhistoryURL}, {5,"Alerts","alerts","a",alertsURL}, {6,"IPs","myinfrastructure","i",myinfrastructureURL\_IP}, {7,"Domains","myinfrastructure","d",myinfrastructureURL\_domain}, {8,"Portfolio Companies","portfolio","a",portfolioURL}, {9,"Portfolio Risk Vector History","rvhistoryport","a",rvhistoryportURL}, {10,"Portfolio Risk Vector Latest","rvlatestport","a",rvlatestportURL}, {11,"Folders","folders","a",foldersURL}, {12,"Tiers", "tiers","a",tiersURL}, {13,"Portfolio Ratings History", "ratings","a",ratingsURL}, {14,"Breaches","breaches","a",breachesURL}, {15,"Rating Changes", "ratingch","a",ratingchURL}, {16,"Industries", "industry","a",industryURL}, {17,"Infrastructure", "infrastructure","a",infrastructureURL}, {18,"Assets", "assets","a",assetsURL}, // VRM Implementation {19,"Vendors", "vendors","a",allVendorsURL}, {19,"Vendors Details", "vendors","d","dynamic"}, {24,"Vendor Rating", "vendors","r",vendorsRatingURL}, {25,"Vendor Information", "vendors","i",vendorsInfoURL}, {26,"Vendor Contacts", "contacts","v",vendorsContactsURL}, {27,"Internal Contacts", "contacts","i",internalContactsURL}, {28,"Vendor Findings", "vfindings","a",allVendorFindingsURL}, {29,"Vendor Findings Details", "vfindings","d",findingDetailsURL}, {30,"Vendor Internal Questionnaires", "iquestionnaires","a",internalQuestionnairesURL}, {31,"Vendor Internal Questionnaire Categories", "iquestionnaires","c",questionnairesCategoriesURL}, {32,"Vendor Internal Questionnaire Questions", "iquestionnaires","q",internalQuestionnairesInfoQURL}, {33,"Vendor Internal Questionnaire Answers", "iquestionnaires","w",internalQuestionnairesInfoAURL}, {34,"Vendor Internal Questionnaire Progress", "iquestionnaires","p",internalQuestionnairesProgressURL}, {35,"Vendor External Questionnaires", "equestionnaires","a",externalQuestionnairesURL}, {36,"Vendor External Questionnaire Categories", "equestionnaires","c",questionnairesCategoriesURL}, {37,"Vendor External Questionnaire Questions", "equestionnaires","q",externalQuestionnairesInfoQURL}, {38,"Vendor External Questionnaire Answers", "equestionnaires","w",externalQuestionnairesInfoAURL}, {39,"Vendor External Questionnaire Progress", "equestionnaires","p",externalQuestionnairesProgressURL}, {40,"Vendors Requirements Progress", "requirements","a",requirementsURL}, {41,"Vendors Requirements Details", "requirements","d",requirementsDetailsURL}, {42,"Lifecycle Stages", "lifecycle","a",lifeCycleURL} }); |

Code Block 1 RootEntities

Each entity in the RootEntities table includes the following fields:

* **Entity Order**: The sequence number for display or processing order.
* **Entity Name**: The display name of the entity on the navigation table.
* **Endpoint**: The API endpoint or category associated with the entity.
* **Flag**: A shorthand character to indicate the type of data or operation.
* **URL**: The endpoint's URL or dynamic generation indicator ("dynamic" for runtime construction).

Each entity is eventually translated into a table.

## Schemas

The SchemaTable defines the structure (schema) for entities available in the navigation table. It specifies the **field names** and their **data types** for each entity, which helps in standardizing data processing, validation, and display. You have one schema for each entity in the Entities list.

The SchemaTable contains the following fields:

* **Entity**: Name of the entity for which the schema applies (e.g., "Details", "Ratings History").
* **SchemaTable**: A table of attribute definitions for the entity, listing:
	+ **Name**: Attribute or column name.
	+ **Type**: Data type of the attribute.

You can see more details about the schemas of each API on its technical doc page. Here for the Security Ratings API and here for the Vendor Ratings Management API.

Additional Notes:

1. **Expandability**: New entities and schemas can be added as needed by defining their attributes and types in the SchemaTable.
2. **Type Mapping**:
	* type text: Represents string values.
	* type number: Represents integer or floating-point numeric values.
	* type date: Represents date values.
	* type logical: Represents Boolean values (true/false).
	* type list: Represents a list of items.
	* type record: Represents a nested structure with key-value pairs.
3. **Usage**: This schema table is referenced during API calls to parse, validate, and display the data correctly in tools like Power BI.

## Get Entity and Apply Schema

The GetEntity function is a dynamic and extensible method used to retrieve data from various API endpoints and apply the appropriate schema for data processing in Power Query. It acts as a universal data-fetching function, allowing seamless integration with multiple endpoints by dynamically determining the schema and applying the correct data extraction logic.

|  |
| --- |
| // Obtain an endpoint entity and apply schemaGetEntity = (url as text, endpoint as text, flag as text, company\_guid as text, entity\_name as text, optional affects\_rating\_findings as text) as table =>  let schemaTable = GetSchemaForEntity(entity\_name), result = if endpoint = "company" and flag = "d" then BitSightSecurityRatingsCompanyDetails.Feed(url, endpoint, flag, company\_guid, schemaTable) else if endpoint = "company" and flag = "h" then BitSightSecurityRatingsCompanyRatings.Feed(url, endpoint, flag, company\_guid, schemaTable) else if endpoint = "company" and flag = "g" then BitSightSecurityRatingsCompanyGrades.Feed(url, endpoint, flag, company\_guid, schemaTable) else if endpoint = "alerts" or endpoint = "rvhistory" or endpoint = "findings" or endpoint = "portfolio" or endpoint = "rvhistoryport" or endpoint= "rvlatestport" or endpoint= "ratingch" or endpoint = "findingsport" or endpoint = "summaries" or endpoint = "infrastructure" or endpoint = "assets" then  PaginatedResult(url, company\_guid, endpoint, affects\_rating\_findings, schemaTable) else if endpoint = "myinfrastructure" then  BitSightSecurityRatingsMyInfrastructure.Feed(url, flag, company\_guid, schemaTable) else if endpoint = "folders" then  BitSightSecurityRatingsFolders.Feed(url, flag, company\_guid, schemaTable) else if endpoint = "tiers" then  BitSightSecurityRatingsTiers.Feed(url, flag, company\_guid, schemaTable) else if endpoint = "ratings" then  BitSightSecurityRatingsRating.Feed(url, flag, company\_guid, schemaTable) else if endpoint = "breaches" then  BitSightSecurityRatingsBreaches.Feed(url, flag, company\_guid, schemaTable) else if endpoint = "industry" then  BitSightSecurityRatingsIndustry.Feed(url, flag, company\_guid, schemaTable) // VRM API else if endpoint = "vendors" and flag = "a" then  BitSightSecurityAllVendors.Feed(url, flag, company\_guid, schemaTable) else if endpoint = "vendors" and flag = "d" then  BitSightSecurityVendorsDetails.Feed(url, flag, company\_guid, schemaTable) else if endpoint = "vendors" and flag = "r" then  BitSightSecurityVendorRating.Feed(url, flag, company\_guid, schemaTable) else if endpoint = "vendors" and flag = "i" then  BitSightSecurityVendorInfo.Feed(url, flag, company\_guid, schemaTable) else if endpoint = "contacts" and flag = "v" then  BitSightSecurityVendorContacts.Feed(url, flag, company\_guid, schemaTable) else if endpoint = "contacts" and flag = "i" then  BitSightSecurityInternalContacts.Feed(url, flag, company\_guid, schemaTable) else if endpoint = "vfindings" and flag = "a" then  BitSightSecurityFindings.Feed(url, flag, company\_guid, schemaTable) else if endpoint = "vfindings" and flag = "d" then  BitSightSecurityFindingsDetails.Feed(url, flag, company\_guid, schemaTable) else if endpoint = "iquestionnaires" and flag = "a" then  BitSightSecurityInternalQuestionnaires.Feed(url, flag, company\_guid, schemaTable) else if endpoint = "iquestionnaires" and flag = "c" then  BitSightSecurityInternalQuestionnairesCategories.Feed(url, flag, company\_guid, schemaTable) else if endpoint = "iquestionnaires" and flag = "q" then  BitSightSecurityInternalQuestionnairesQuestions.Feed(url, flag, company\_guid, schemaTable) else if endpoint = "iquestionnaires" and flag = "w" then  BitSightSecurityInternalQuestionnairesAnswers.Feed(url, flag, company\_guid, schemaTable) else if endpoint = "iquestionnaires" and flag = "p" then  BitSightSecurityInternalQuestionnairesProgress.Feed(url, flag, company\_guid, schemaTable) else if endpoint = "equestionnaires" and flag = "a" then  BitSightSecurityExternalQuestionnaires.Feed(url, flag, company\_guid, schemaTable) else if endpoint = "equestionnaires" and flag = "c" then  BitSightSecurityExternalQuestionnairesCategories.Feed(url, flag, company\_guid, schemaTable) else if endpoint = "equestionnaires" and flag = "q" then  BitSightSecurityExternalQuestionnairesQuestions.Feed(url, flag, company\_guid, schemaTable) else if endpoint = "equestionnaires" and flag = "w" then  BitSightSecurityExternalQuestionnairesAnswers.Feed(url, flag, company\_guid, schemaTable) else if endpoint = "equestionnaires" and flag = "p" then  BitSightSecurityExternalQuestionnairesProgress.Feed(url, flag, company\_guid, schemaTable) else if endpoint = "requirements" and flag = "a" then  BitSightSecurityRequirements.Feed(url, flag, company\_guid, schemaTable) else if endpoint = "requirements" and flag = "d" then  BitSightSecurityRequirementsDetails.Feed(url, flag, company\_guid, schemaTable) else if endpoint = "lifecycle" and flag = "a" then  BitSightSecurityLifecycle.Feed(url, flag, company\_guid, schemaTable) else  error Extension.LoadString("EndpointWithNoDataFunctionError") in result; |

Code Block 2 GetEntity

### Purpose

The GetEntity function:

1. Retrieves data from a specified API endpoint.
2. Applies the schema associated with the entity for validation and consistency.
3. Handles different endpoints and entity-specific parameters dynamically, ensuring versatility and reusability across multiple use cases.

### Function Logic

1. **Determine Schema**:
	* The function begins by retrieving the schema table for the specified entity\_name using GetSchemaForEntity(entity\_name).
2. **Endpoint Selection**:
	* Based on the combination of endpoint and flag, the function routes the request to the appropriate feed function.
	* For endpoints like "company", "alerts", or "findings", the function dynamically identifies the correct feed logic.
3. **Feed Function Execution**:
	* The function calls the appropriate feed (e.g., BitSightSecurityRatingsCompanyDetails.Feed, PaginatedResult) to fetch data.
	* Each feed function uses the schemaTable to validate and transform the data into the expected format.
4. **Error Handling**:
	* If no suitable combination of endpoint and flag is found, the function throws an error: EndpointWithNoDataFunctionError.

## Diagram Flow of Data

The entities that represent tables are declared in an entities list that is used for the navigation table. These are the tables that the user will see when they first connect to the connector and select which tables to load with one or more API calls to the Bitsight APIs.

Once the tables are selected, the connector will proceed to use the functions defined for that specific table/entity. Let's go over the function/data flow of a call that wants to get a table with all the VRM vendor details.



The user starts by selecting the vendor details from the navigation table/menu when first connecting to the connector.

The connector will then proceed to process this entity. The first action is to get the schema related to this entity. The schema tells the connector how many columns and its names and types that will be the resulting table. Then the function proceeds with the API Calls.

# PowerBI Connector for Bitsight - Release 4.1.0

##

* [**Overview**](#scroll-bookmark-24)
* [**Technologies Used**](#scroll-bookmark-25)
* [**Bitsight APIs**](#scroll-bookmark-26)
* [**Architecture Diagrams**](#scroll-bookmark-27)
* [**Certification**](#scroll-bookmark-28)

## Overview

The BitSight Power BI Connector bridges Power BI and BitSight’s API ecosystem, enabling users to seamlessly access, visualize, and analyze security ratings and vendor risk data. By integrating data from both **APIs**, the connector empowers organizations to make data-driven decisions on cybersecurity and vendor risk management.

This development for Bitsight was done in two phases:

1. The first development was to implement the retrieval of data from the Continuous Monitoring API.
2. The second development was to implement the retrieval of data from the Vendors Risk Management API.

These two developments fall under the same connector, i.e. the second development is an extension of the first.

## Technologies Used

Power Query (M Language):

The core technology used to build the connector, handling API requests, data transformations, and loading into Power BI.

Bitsight APIs:

The data comes from two different [bitsight's APIs](https://bitsight.stoplight.io/) (https://bitsight.stoplight.io/) in JSON Format.

Power BI:

The visualization platform that consumes and presents the data retrieved via the connector.

## Bitsight APIs

* **Bitsight Continuous Monitoring API:** Provides security ratings for internal and external companies. The Bitsight API gives access the current security rating for companies that belong to user's portfolio, each industry, and daily ratings for each company and weekly ratings for each industry. Sets a high standard for how Bitsight Security Ratings deliver value to a third-party risk management program and allows to make informed decisions to improve operational workflows.
* **Bitsight API: Bitsight VRM:** Focused on Vendor Risk Management, offering insights into vendor portfolios and risk factors. The Bitsight VRM API provides the vendors data and perform most of the actions available in theBitsight VRM web portal itself. The API documentation for Bitsight VRM is hosted in Stoplight and allows you to make API requests directly from the documentation. You can access this documentation [here](https://bitsight.stoplight.io/docs/VRM-api) (https://bitsight.stoplight.io/docs/VRM-api).

## Architecture Diagrams

The architecture of the proposed solution is described below:



* The Power Query Custom Connector will extract data from Bitsight's plataform through its API
* The end-user needs to place the connector file in a specific folder in his own computer to be able to
use it in Power BI Desktop. Power BI will then connect to data in import mode, and the user
will be able to create reports.
* Once the reports are created, they can be published in Power BI Service, and a data refresh can be
scheduled through Power BI on-premises data gateway.

Here we have a more detailed look at a diagram about the user experience:



## Certification

As a future improvement, it was made possible to make the connector publicly available in Power BI, this required Bitsght to go through the Microsoft connector certification process. You can find more information on this process in the link below:
<https://docs.microsoft.com/en-us/power-query/connectorcertification> (https://docs.microsoft.com/en-us/power-query/connectorcertification)

# Manual Installation of a Power BI Connector - Release 4.1.0

* [Overview](#scroll-bookmark-31)
	+ [Why Manual Installation is Needed?](#scroll-bookmark-32)
		- [Common Use Cases:](#scroll-bookmark-33)
* [Prerequisites](#scroll-bookmark-34)
	+ [Step-by-Step Installation](#scroll-bookmark-35)
		- [1. Enable Custom Connectors in Power BI](#scroll-bookmark-36)
		- [2. Locate the Custom Connectors Folder](#scroll-bookmark-37)
		- [3. Copy the .mez File](#scroll-bookmark-38)
		- [4. Restart Power BI Desktop](#scroll-bookmark-39)
		- [5. Access the Connector](#scroll-bookmark-40)
* [Best Practices](#scroll-bookmark-41)

## Overview

Power BI connectors allow users to access and integrate data from custom or external sources that aren't supported natively in Power BI. If your organization develops a custom connector, such as one for Bitsight APIs, you might need to manually install the connector for testing or deployment. **Please note that bitsight customers should access the connector through the official list of Microsoft certified connectors, and not through manually installing them.**

This guide explains the manual installation process for a Power BI connector (.mez file), detailing the reasons for manual installation and the steps required.

You can see Microsoft's official documentation on this subject [here](https://learn.microsoft.com/en-us/power-bi/connect-data/desktop-connector-extensibility#custom-connectors) (https://learn.microsoft.com/en-us/power-bi/connect-data/desktop-connector-extensibility#custom-connectors)

### Why Manual Installation is Needed?

#### Common Use Cases:

1. **Custom Development**: The connector is still in development or testing and not yet officially deployed or available in AppSource.
2. **Enterprise Use**: The connector is tailored for your company’s specific data sources and not publicly distributed.
3. **Troubleshooting**: Manual installation allows for quick testing of updates without needing a formal deployment process (Microsoft certification)

## Prerequisites

1. **Power BI Desktop**: Ensure you have Power BI Desktop installed (version supporting custom connectors). Download it from [Microsoft's website](https://powerbi.microsoft.com/desktop/) (https://powerbi.microsoft.com/desktop/).
2. **.mez File**: Obtain the .mez file from Xpand IT or download it from a trusted source within your organization.

### Step-by-Step Installation

#### 1. Enable Custom Connectors in Power BI

Power BI does not enable custom connectors by default. Follow these steps to activate them:

* Open **Power BI Desktop**.
* Navigate to **File > Options and Settings > Options**.
* Under **Global > Security**, check **Allow any extension to load without validation or warning** (or a similar option).

|  |
| --- |
| **Note:** This setting is necessary to allow Power BI to load your custom connector. Be cautious when using untrusted connectors. |

#### 2. Locate the Custom Connectors Folder

Power BI uses a designated folder to store custom connectors. Determine the appropriate folder for your setup:

* **Default Path for Windows**:
	+ User-specific installation:
	C:\Users\<YourUsername>\Documents\Power BI Desktop\Custom Connectors\
	+ Shared for all users (requires admin privileges):
	C:\Program Files\Microsoft Power BI Desktop\Custom Connectors\
* If these folders don't exist, you may need to create them manually.

#### 3. Copy the .mez File

* Place the .mez file into the appropriate **Custom Connectors** folder:
	+ For user-specific: Documents\Power BI Desktop\Custom Connectors\
	+ For all users: Program Files\Microsoft Power BI Desktop\Custom Connectors\

#### 4. Restart Power BI Desktop

* Close Power BI Desktop if it’s open.
* Reopen the Power BI Desktop to allow the connector to load. Upon restart, the custom connector should now be available under the **Get Data** menu.

#### 5. Access the Connector

* Go to **Get Data > More > Other**.
* Look for the connector by name (e.g., "Bitsight Connector").
* Select the connector and follow any prompts (e.g., entering your API token key).

## Best Practices

* **File Verification**: Only install .mez files from trusted sources to avoid security risks.
* **Documentation**: Keep a record of installed connectors, their sources, and their purposes for audit purposes.
* **Regular Updates**: Check with your development team for updates to ensure compatibility with newer versions of Power BI.

By following this tutorial, you can successfully install and use a custom Power BI connector to access data from your organization's specific APIs or data sources. This process is simple yet powerful, enabling tailored data integration and enhanced reporting capabilities.

# Bitsight Continuous Monitoring API - Release 4.1.0

* [Introduction](#scroll-bookmark-44)
* [Helpful Variables](#scroll-bookmark-45)
* [Endpoints for Continuous Monitoring API](#scroll-bookmark-46)
* [Pagination](#scroll-bookmark-47)
	+ [What is Pagination?](#scroll-bookmark-48)
		- [How Pagination Works in This Code](#scroll-bookmark-49)
	+ [Explanation of the Code](#scroll-bookmark-50)
		- [1. GetAllPagesByNextLink Function](#scroll-bookmark-51)
		- [2. GetNextLink Function](#scroll-bookmark-52)
		- [3. Table.GenerateByPage Function](#scroll-bookmark-53)
	+ [Summary](#scroll-bookmark-54)

## Introduction

The Bitsight API allows developers to build applications around the Bitsight data or integrate Bitsight Security Ratings into their existing systems. The API returns JSON which is then accessible to the user through the Power BI Connector. You can read more about this API [here](https://help.bitsighttech.com/hc/en-us/articles/231872628-API-Documentation-Overview) (https://help.bitsighttech.com/hc/en-us/articles/231872628-API-Documentation-Overview).

This was the first implementation for Bitsight and the main difference between this implementation and the Vendors Ratings Management API implementation is:

* Different API
* Use of pagination when the amount of data is big

## Helpful Variables

Here’s a table explaining the endpoints and variables used in this implementation of the API for Continuous Monitoring:

|  |  |  |
| --- | --- | --- |
| Variable | Description | Example/Details |
| page\_size | Specifies the number of items retrieved per API call. | Value: 250 |
| BaseUrl | Base URL for the BitSight Normal API. | <https://api.bitsighttech.com/> (https://api.bitsighttech.com/) |
| rvhistory\_period | Specifies the time period for risk vector history data. | Value: monthly |
| rvlatest\_period | Specifies the latest risk vector period. | Value: latest |
| myInfrastructure\_format | Defines the format for infrastructure reports. | Value: csv |
| myInfrastructure\_type\_IP | Specifies the type of infrastructure report as IP-based. | Value: ip |
| myInfrastructure\_type\_domain | Specifies the type of infrastructure report as domain based. | Value: domain |
| helpURL | Link to the help document. | [BitSight Help Terms](https://overviewdocs.bitsighttech.com/BitSightForPowerBIAppTerms.pdf) (https://overviewdocs.bitsighttech.com/BitSightForPowerBIAppTerms.pdf) |
| default\_affects\_rating\_findings | Filters’ findings based on whether they affect ratings. | Value: all |

## Endpoints for Continuous Monitoring API

There are 18 different tables being loaded to the powerBI connector using this API, here are the endpoints being used.

|  |  |  |
| --- | --- | --- |
| Endpoint | Description | URL |
| companyURL | Retrieves company data. | <https://api.bitsighttech.com/ratings/v1/companies/> (https://api.bitsighttech.com/ratings/v1/companies/) |
| findingsURL | Retrieves findings for a company with additional remediation history and attributed companies. | <https://api.bitsighttech.com/ratings/v1/companies/company_guid/findings?expand=remediation_history,attributed_companies&limit=250> (https://api.bitsighttech.com/ratings/v1/companies/company\_guid/findings?expand=remediation\_history,attributed\_companies&limit=250) |
| rvhistoryURL | Fetches risk vector grades over a historical period for a company. | <https://api.bitsighttech.com/v1/portfolio/risk-vectors/grades?period=monthly&limit=250&company.guid=company_guid> (https://api.bitsighttech.com/v1/portfolio/risk-vectors/grades?period=monthly&limit=250&company.guid=company\_guid) |
| alertsURL | Retrieves alerts for a company or portfolio | <https://api.bitsighttech.com/ratings/v2/alerts> (https://api.bitsighttech.com/ratings/v2/alerts) |
| myinfrastructureURL\_IP | Fetches IP-based infrastructure data for a company. | <https://api.bitsighttech.com/ratings/v1/companies/company_guid/reports/infrastructure?format=csv&type=ip> (https://api.bitsighttech.com/ratings/v1/companies/company\_guid/reports/infrastructure?format=csv&type=ip) |
| myinfrastructureURL\_domain | Fetches domain-based infrastructure data for a company. | <https://api.bitsighttech.com/ratings/v1/companies/company_guid/reports/infrastructure?format=csv&type=domain> (https://api.bitsighttech.com/ratings/v1/companies/company\_guid/reports/infrastructure?format=csv&type=domain) |
| guidUrl | Retrieves the GUID for the current user. | <https://api.bitsighttech.com/v1/users/current> (https://api.bitsighttech.com/v1/users/current) |
| portfolioURL | Fetches data about companies in the user’s portfolio. | <https://api.bitsighttech.com/ratings/v2/portfolio/> (https://api.bitsighttech.com/ratings/v2/portfolio/) |
| rvlatestportURL | Fetches the latest risk vector grades for a portfolio. | <https://api.bitsighttech.com/v1/portfolio/risk-vectors/grades?period=latest&limit=250> (https://api.bitsighttech.com/v1/portfolio/risk-vectors/grades?period=latest&limit=250) |
| foldersURL | Retrieves folder data for organizing entities. | <https://api.bitsighttech.com/v1/folders> (https://api.bitsighttech.com/v1/folders) |
| tiersURL | Retrieves tier-related data for portfolio companies. | <https://api.bitsighttech.com/v1/tiers> (https://api.bitsighttech.com/v1/tiers) |
| ratingsURL | Retrieves ratings data for portfolio companies. | <https://api.bitsighttech.com/v1/portfolio/ratings?period=monthly> (https://api.bitsighttech.com/v1/portfolio/ratings?period=monthly) |
| breachesURL | Fetches breach data for portfolio companies. | <https://api.bitsighttech.com/v1/portfolio/breaches> (https://api.bitsighttech.com/v1/portfolio/breaches) |
| ratingchURL | Fetches rating change insights. | <https://api.bitsighttech.com/v1/insights/rating_changes> (https://api.bitsighttech.com/v1/insights/rating\_changes) |
| industryURL | Retrieves industry-specific data. | <https://api.bitsighttech.com/ratings/v1/industries> (https://api.bitsighttech.com/ratings/v1/industries) |
| infrastructureURL | Fetches infrastructure data for a company. | <https://api.bitsighttech.com/ratings/v1/companies/company_guid/infrastructure> (https://api.bitsighttech.com/ratings/v1/companies/company\_guid/infrastructure) |
| assetsURL | Retrieves asset details for a company. | <https://api.bitsighttech.com/ratings/v1/companies/company_guid/assets?expand=tag_details> (https://api.bitsighttech.com/ratings/v1/companies/company\_guid/assets?expand=tag\_details) |

## Pagination

### What is Pagination?

Pagination is a technique used in APIs to manage large sets of data by breaking them into smaller, more manageable chunks (pages). Instead of returning all data at once, the API provides a limited number of records per request along with a next link that can be used to retrieve subsequent pages.

#### How Pagination Works in This Code

1. **Initial Request:** The first request is made to the API with the base URL.
2. **Retrieve Data:** The API returns a response containing:
	* A list of records (data).
	* A next link in the metadata, pointing to the next page.
3. **Follow the Next Link:** The function checks if the next link is present:
	* If yes, it makes another request using the next link.
	* If not, it stops fetching more pages.
4. **Combine Data:** The process repeats until all pages are retrieved, and the data from each page is combined into a single table.

### Explanation of the Code

#### 1. GetAllPagesByNextLink Function

This function handles pagination by repeatedly following the next link in the API response and fetching new pages of data.

**Steps in the function:**

* It uses Table.GenerateByPage to create an iterative process that keeps calling API pages until there are no more results.
* Determines the correct API endpoint to use based on the endpoint parameter.
* Calls the appropriate function to fetch a specific type of paginated data:
	+ Findings (GetPageFindings)
	+ Alerts (GetPageAlerts)
	+ Rating History (GetPageRVHistory)
	+ Portfolio (GetPagePortfolio)
	+ Infrastructure (GetPageInfrastructure)
	+ Assets (GetPageAssets)
	+ And more, depending on the requested endpoint.
* Stop fetching when there is no next link in the response.

|  |
| --- |
| // Obtain Links - PaginationGetAllPagesByNextLink = (url as text, company\_guid as text, affects\_rating\_findings as text, optional endpoint as text, optional schema as table) as table => Table.GenerateByPage((previous) =>  let // if previous is null, then this is our first page of data nextLink = if (previous = null) then url else Value.Metadata(previous)[NextLink]?, // if NextLink was set to null by the previous call, we know we have no more data page = if (nextLink <> null) then  if endpoint = "findings" then  GetPageFindings(nextLink, company\_guid, affects\_rating\_findings, schema) else if endpoint = "alerts" then  GetPageAlerts(nextLink, schema) else if endpoint = "rvhistory" then  GetPageRVHistory(nextLink, company\_guid, schema) else if endpoint = "portfolio" then GetPagePortfolio(nextLink, schema) else if endpoint = "rvhistoryport" then GetPageRVHistoryPort(nextLink,schema) else if endpoint = "rvlatestport" then GetPageRVLatestPort(nextLink, schema) else if endpoint = "ratingch" then GetPageRatingCH(nextLink, schema) else if endpoint = "infrastructure" then GetPageInfrastructure(nextLink, company\_guid, schema) else if endpoint = "assets" then GetPageAssets(nextLink, company\_guid, schema) else error Extension.LoadString("EndpointWithNoPaginationError") else null in page );  |

#### 2. GetNextLink Function

This function extracts the next link from the API response metadata, allowing the pagination process to continue.

**Steps in the function:**

* Extracts the next URL from the links section of the API response.
* If there is no next link, it returns null, signaling that no more pages are available.

|  |
| --- |
| // Common - Obtain Next Link - PaginationGetNextLink = (response) as nullable text =>  let next\_url = response[links][next] in next\_url; |

#### 3. Table.GenerateByPage Function

This function is responsible for handling the iteration process.

**Steps in the function:**

* Calls getNextPage(null) to fetch the first page.
* Use List.Generate to repeatedly call getNextPage until there are no more pages.
* Convert the list of pages into a table and expand all columns from the first page to standardize the output.
* If no data is retrieved, it returns an empty table.

|  |
| --- |
| // The getNextPage function takes a single argument and is expected to return a nullable tableTable.GenerateByPage = (getNextPage as function) as table => let  listOfPages = List.Generate( () => getNextPage(null), // get the first page of data (lastPage) => lastPage <> null, // stop when the function returns null (lastPage) => getNextPage(lastPage) // pass the previous page to the next function call ), // concatenate the pages together tableOfPages = Table.FromList(listOfPages, Splitter.SplitByNothing(), {"Column1"}), firstRow = tableOfPages{0}? in // if we didn't get back any pages of data, return an empty table // otherwise set the table type based on the columns of the first page if (firstRow = null) then Table.FromRows({}) else  Value.ReplaceType( Table.ExpandTableColumn(tableOfPages, "Column1", Table.ColumnNames(firstRow[Column1])), Value.Type(firstRow[Column1]) ); |

### Summary

* **Pagination** helps in managing large data sets by retrieving them in chunks.
* GetAllPagesByNextLink fetches all pages of data dynamically based on the provided endpoint.
* GetNextLink extracts the next link from API responses.
* Table.GenerateByPage handles the iteration, making API calls until no more data is available.

This approach ensures efficient data retrieval, reducing memory usage and improving performance by processing smaller data sets at a time.

# Bitsight Vendor Ratings Management API - Release 4.1.0

* [Introduction](#scroll-bookmark-57)
* [Power BI Data Modelling](#scroll-bookmark-58)
* [Entities](#scroll-bookmark-59)
	+ [Endpoints for Vendor Rating Management API](#scroll-bookmark-60)
* [Code explanation](#scroll-bookmark-61)
	+ [All Vendors](#scroll-bookmark-62)
	+ [Vendor Details](#scroll-bookmark-63)
	+ [Vendor Rating](#scroll-bookmark-64)
	+ [Vendor Info](#scroll-bookmark-65)
	+ [Vendor Contacts](#scroll-bookmark-66)
	+ [Vendor Internal Contacts](#scroll-bookmark-67)
	+ [Vendor Findings](#scroll-bookmark-68)
	+ [Findings Details](#scroll-bookmark-69)
	+ [Internal Questionnaires:](#scroll-bookmark-70)
	+ [Internal Questionnaires Categories](#scroll-bookmark-71)
	+ [Internal Questionnaires Questions](#scroll-bookmark-72)
	+ [Internal Questionnaires Answers](#scroll-bookmark-73)
	+ [Internal Questionnaires Progress](#scroll-bookmark-74)
	+ [External Questionnaires](#scroll-bookmark-75)
	+ [External Questionnaires Categories](#scroll-bookmark-76)
	+ [External Questionnaires Answers](#scroll-bookmark-77)
	+ [External Questionnaires Questions](#scroll-bookmark-78)
	+ [External Questionnaires Progress](#scroll-bookmark-79)
	+ [Vendor Requirements](#scroll-bookmark-80)
	+ [Vendor Requirements Details](#scroll-bookmark-81)
	+ [Vendor Lifecyle](#scroll-bookmark-82)
* [Performance Issues](#scroll-bookmark-83)

## Introduction

The VRM API is designed to manage and retrieve data related to vendor risk and compliance within an organization. It provides endpoints for fetching vendor details, ratings, contacts, findings, questionnaires, requirements, and lifecycle stages. Each endpoint is tailored for specific use cases, with flexible parameters to handle various scenarios.

The API returns JSON which is then accessible to the user through the Power BI Connector. You can read more about this API [here](https://bitsight.stoplight.io/docs/VRM-api/branches/main/seipzbygt03mg-getting-started-with-the-vrm-api) (https://bitsight.stoplight.io/docs/VRM-api/branches/main/seipzbygt03mg-getting-started-with-the-vrm-api).

Take notice of the [Performance Issues](#scroll-bookmark-83) section before using the Connector

## Power BI Data Modelling

In Power BI, you can create a relationship to create a logical connection between different tables. A relationship enables Power BI to connect tables to one another so that you can create visuals and reports. Once you have received the data using the Bitsight PBI Connector, the user needs to create some of these relationships in PowerBI. The image below is an example of each table connected by their respective keys (click to see larger):



## Entities

There is a root entities list declared in the code for each table that the user will be able to choose from the connector from the Navigation Table.

Below is a detailed description of the entities and their respective endpoints and schemas.

### Endpoints for Vendor Rating Management API

|  |  |  |
| --- | --- | --- |
| Endpoint | Description | URL |
| All Vendors     | Retrieves a list of all vendors.     | <https://service.bitsighttech.com/customer-api/vrm/v1/vendors/query> (https://service.bitsighttech.com/customer-api/vrm/v1/vendors/query) |
| Monitored Vendor Details   | Retrieves details about a monitored vendor.     | [https://service.bitsighttech.com/customer-api/vrm/v1/vendors/monitored/{vendor\_guid](https://service.bitsighttech.com/customer-api/vrm/v1/vendors/monitored/%7Bvendor_guid) (https://service.bitsighttech.com/customer-api/vrm/v1/vendors/monitored/{vendor\_guid)} |
| Managed Vendor Details | Retrieves details about a managed vendor.     | [https://service.bitsighttech.com/customer-api/vrm/v1/vendors/managed/{vendor\_guid](https://service.bitsighttech.com/customer-api/vrm/v1/vendors/managed/%7Bvendor_guid) (https://service.bitsighttech.com/customer-api/vrm/v1/vendors/managed/{vendor\_guid)} |
| Vendor Rating     | Retrieves rating information for a specific vendor.     | [https://service.bitsighttech.com/customer-api/vrm/v1/companies/{bst\_entity\_guid}/rating](https://service.bitsighttech.com/customer-api/vrm/v1/companies/%7Bbst_entity_guid%7D/rating) (https://service.bitsighttech.com/customer-api/vrm/v1/companies/{bst\_entity\_guid}/rating) |
| Vendor Info     | Fetches basic information about a specific vendor.     | [https://service.bitsighttech.com/customer-api/vrm/v1/company/{vendor\_guid](https://service.bitsighttech.com/customer-api/vrm/v1/company/%7Bvendor_guid) (https://service.bitsighttech.com/customer-api/vrm/v1/company/{vendor\_guid)} |
| Vendor Contacts     | Retrieves external contact details for a vendor.     | [https://service.bitsighttech.com/customer-api/vrm/v1/vendors/{vendor\_guid}/contacts](https://service.bitsighttech.com/customer-api/vrm/v1/vendors/%7Bvendor_guid%7D/contacts) (https://service.bitsighttech.com/customer-api/vrm/v1/vendors/{vendor\_guid}/contacts) |
| Internal Contacts     | Retrieves internal contact details for a vendor.     | [https://service.bitsighttech.com/customer-api/vrm/v1/vendors/{vendor\_guid}/internal-contacts](https://service.bitsighttech.com/customer-api/vrm/v1/vendors/%7Bvendor_guid%7D/internal-contacts) (https://service.bitsighttech.com/customer-api/vrm/v1/vendors/{vendor\_guid}/internal-contacts) |
| Internal Questionnaires     | Lists all internal questionnaires for a specific vendor.     | [https://service.bitsighttech.com/customer-api/vrm/v1/vendors/{vendor\_guid}/internal\_questionnaires](https://service.bitsighttech.com/customer-api/vrm/v1/vendors/%7Bvendor_guid%7D/internal_questionnaires) (https://service.bitsighttech.com/customer-api/vrm/v1/vendors/{vendor\_guid}/internal\_questionnaires) |
| Internal Questionnaire Info  | Fetches categories about a specific internal questionnaire of a specific vendor | [https://service.bitsighttech.com/customer-api/vrm/v1/vendors/{vendor\_guid}/internal\_questionnaires/{questionnaire\_guid](https://service.bitsighttech.com/customer-api/vrm/v1/vendors/%7Bvendor_guid%7D/internal_questionnaires/%7Bquestionnaire_guid) (https://service.bitsighttech.com/customer-api/vrm/v1/vendors/{vendor\_guid}/internal\_questionnaires/{questionnaire\_guid)} |
| Internal Questionnaire Questions | Retrieves questions within a specific category of an internal questionnaire of a specific vendor | [https://service.bitsighttech.com/customer-api/vrm/v1/vendors/{vendor\_guid}/internal\_questionnaires/categories/{category\_guid}/questions](https://service.bitsighttech.com/customer-api/vrm/v1/vendors/%7Bvendor_guid%7D/internal_questionnaires/categories/%7Bcategory_guid%7D/questions) (https://service.bitsighttech.com/customer-api/vrm/v1/vendors/{vendor\_guid}/internal\_questionnaires/categories/{category\_guid}/questions) |
| Internal Questionnaire Answers | Retrieves answers within a specific category of an internal questionnaire of a specific vendor | [https://service.bitsighttech.com/customer-api/vrm/v1/vendors/{vendor\_guid}/internal\_questionnaires/categories/{category\_guid}/answers](https://service.bitsighttech.com/customer-api/vrm/v1/vendors/%7Bvendor_guid%7D/internal_questionnaires/categories/%7Bcategory_guid%7D/answers) (https://service.bitsighttech.com/customer-api/vrm/v1/vendors/{vendor\_guid}/internal\_questionnaires/categories/{category\_guid}/answers) |
| Internal Questionnaire Progress | Tracks progress on a specific internal questionnaire.     | [https://service.bitsighttech.com/customer-api/vrm/v1/vendors/{vendor\_guid}/internal\_questionnaires/{questionnaire\_guid}/progress](https://service.bitsighttech.com/customer-api/vrm/v1/vendors/%7Bvendor_guid%7D/internal_questionnaires/%7Bquestionnaire_guid%7D/progress) (https://service.bitsighttech.com/customer-api/vrm/v1/vendors/{vendor\_guid}/internal\_questionnaires/{questionnaire\_guid}/progress) |
| All External Questionnaires     | Retrieves a list of all external questionnaires.     | <https://service.bitsighttech.com/customer-api/vrm/v1/questionnaires/optional> (https://service.bitsighttech.com/customer-api/vrm/v1/questionnaires/optional) |
| Questionnaire Categories | Retrieves categories for a specific questionnaire (internal or external).     | [https://service.bitsighttech.com/customer-api/vrm/v1/questionnaires/{questionnaire\_guid](https://service.bitsighttech.com/customer-api/vrm/v1/questionnaires/%7Bquestionnaire_guid) (https://service.bitsighttech.com/customer-api/vrm/v1/questionnaires/{questionnaire\_guid)} |
| External Questionnaires | Lists all external questionnaires for a specific vendor.     | [https://service.bitsighttech.com/customer-api/vrm/v1/vendors/{vendor\_guid}/questionnaires](https://service.bitsighttech.com/customer-api/vrm/v1/vendors/%7Bvendor_guid%7D/questionnaires) (https://service.bitsighttech.com/customer-api/vrm/v1/vendors/{vendor\_guid}/questionnaires) |
| External Questionnaire Info | Fetches details about a specific external questionnaire.     | [https://service.bitsighttech.com/customer-api/vrm/v1/vendors/{vendor\_guid}/questionnaires/{questionnaire\_guid](https://service.bitsighttech.com/customer-api/vrm/v1/vendors/%7Bvendor_guid%7D/questionnaires/%7Bquestionnaire_guid) (https://service.bitsighttech.com/customer-api/vrm/v1/vendors/{vendor\_guid}/questionnaires/{questionnaire\_guid)} |
| External Questionnaire Questions | Retrieves questions within a specific category of an external questionnaire of a specific vendor. | [https://service.bitsighttech.com/customer-api/vrm/v1/vendors/{vendor\_guid}/questionnaires/categories/{category\_guid}/questions](https://service.bitsighttech.com/customer-api/vrm/v1/vendors/%7Bvendor_guid%7D/questionnaires/categories/%7Bcategory_guid%7D/questions) (https://service.bitsighttech.com/customer-api/vrm/v1/vendors/{vendor\_guid}/questionnaires/categories/{category\_guid}/questions) |
| External Questionnaire Answers | Retrieves answers within a specific category of an external questionnaire of a specific vendor. | [https://service.bitsighttech.com/customer-api/vrm/v1/vendors/{vendor\_guid}/questionnaires/categories/{category\_guid}/answers](https://service.bitsighttech.com/customer-api/vrm/v1/vendors/%7Bvendor_guid%7D/questionnaires/categories/%7Bcategory_guid%7D/answers) (https://service.bitsighttech.com/customer-api/vrm/v1/vendors/{vendor\_guid}/questionnaires/categories/{category\_guid}/answers) |
| External Questionnaire Progress | Tracks progress on a specific external questionnaire of a specific vendor. | [https://service.bitsighttech.com/customer-api/vrm/v1/vendors/{vendor\_guid}/questionnaires/{questionnaire\_guid}/progress](https://service.bitsighttech.com/customer-api/vrm/v1/vendors/%7Bvendor_guid%7D/questionnaires/%7Bquestionnaire_guid%7D/progress) (https://service.bitsighttech.com/customer-api/vrm/v1/vendors/{vendor\_guid}/questionnaires/{questionnaire\_guid}/progress) |
| All Vendor Findings | Queries all findings associated with vendors.     | <https://service.bitsighttech.com/customer-api/vrm/v1/vendors/findings/query> (https://service.bitsighttech.com/customer-api/vrm/v1/vendors/findings/query) |
| Finding Details     | Retrieves details for a specific finding.     | [https://service.bitsighttech.com/customer-api/vrm/v1/vendors/findings/{finding\_guid](https://service.bitsighttech.com/customer-api/vrm/v1/vendors/findings/%7Bfinding_guid) (https://service.bitsighttech.com/customer-api/vrm/v1/vendors/findings/{finding\_guid)} |
| Requirements | Lists all requirements related to all vendors.     | <https://service.bitsighttech.com/customer-api/vrm/v1/vendors/requirements> (https://service.bitsighttech.com/customer-api/vrm/v1/vendors/requirements) |
| Requirement Details     | Fetches details for a specific requirement of a vendor.     | [https://service.bitsighttech.com/customer-api/vrm/v1/vendors/{vendor\_guid}/requirements/{requirement\_guid](https://service.bitsighttech.com/customer-api/vrm/v1/vendors/%7Bvendor_guid%7D/requirements/%7Brequirement_guid) (https://service.bitsighttech.com/customer-api/vrm/v1/vendors/{vendor\_guid}/requirements/{requirement\_guid)} |
| Lifecycle Stages | Lists the lifecycle stages for vendors. | <https://service.bitsighttech.com/customer-api/vrm/v1/life-cycle-stages> (https://service.bitsighttech.com/customer-api/vrm/v1/life-cycle-stages) |

Notes: **Dynamic URL Handling**: Some entities (e.g., Vendor Details) rely on dynamically constructed URLs.

## Code explanation

### All Vendors

|  |
| --- |
| BitSightSecurityAllVendors.Feed = (url as text, endpoint as text, flag as text, optional schema as table) => let source = Web.Contents(url, [ Headers = DefaultVRMRequestHeaders, Content = Json.FromValue([]) ]), json = Json.Document(source), resultsList = json[results], resultsTable = Table.FromList(resultsList, Splitter.SplitByNothing(), null, null, ExtraValues.Error), table\_fields = Record.FieldNames(Table.FirstValue(resultsTable, [Empty = null])), expanded = Table.ExpandRecordColumn(resultsTable, "Column1", table\_fields), #"Added Custom" = Table.AddColumn(expanded, "type", each if [is\_managed] = true then "Managed" else "Monitored"), finalTable = Table.RemoveColumns(#"Added Custom",{"is\_managed"}), withSchema = if schema <> null then SchemaTransformTable(finalTable, schema) else finalTable in withSchema; |

Steps in the Function:

1. **Fetch Data from the API** - The Web.Contents function makes a POST request to the given url with DefaultVRMRequestHeaders and an empty JSON body. The response contains the vendor data.
2. **Parse JSON Response** - The response is parsed from JSON format. The results field, which contains the list of vendors, is extracted into resultsList.
3. **Convert List to Table** - The list of results is converted into a Power Query table, with each list item as a record in the table.
4. **Extract Table Fields** - The field names of the first record in the results table are extracted to ensure all records can be expanded. The Column1 is then expanded to include all its fields in separate columns.
5. **Add "type" Column** - A new column named type is added, indicating whether each vendor is "Managed" or "Monitored" based on the value of the is\_managed field.
6. **Remove *is\_managed* Column** - The is\_managed column is removed, as its information is now represented in the type column.
7. **Apply Schema Transformation** (Optional) - If a schema is provided, the SchemaTransformTable function applies the schema to the finalTable. Otherwise, the table is returned as is

Complexity: One call to get All Vendors

### Vendor Details

Function 1: GetVendorRating: This function retrieves the rating details for a specific vendor by replacing the {bst\_entity\_guid} placeholder in the URL with the vendor's unique identifier and fetching data from the resulting API endpoint.

Function 2: BitSightSecurityVendorsDetails.Feed:
This function retrieves and combines detailed information for all vendors, separating them into "Managed" and "Monitored" categories.

|  |
| --- |
| GetVendorDetails = (url as text, vendor\_guid as text) => let details\_url = Text.Replace(url, "{vendor\_guid}", vendor\_guid), source = Web.Contents(details\_url, [Headers = DefaultVRMRequestHeaders]), json = Json.Document(source), detailsRecord = Record.AddField(json, "vendor\_guid", vendor\_guid) in detailsRecord; BitSightSecurityVendorsDetails.Feed = (url as text, endpoint as text, flag as text, optional schema as table) => let allVendors = BitSightSecurityAllVendors.Feed(allVendorsURL, endpoint, flag, null), vendorDetails = Table.SelectColumns(allVendors, {"vendor\_guid", "type"}),  // Transform each row to retrieve vendor details detailsList = List.Transform( Table.ToRecords(vendorDetails),  each  if [type] = "Managed" then GetVendorDetails(managedVendorsDetailsURL, [vendor\_guid]) else GetVendorDetails(monitoredVendorsDetailsURL, [vendor\_guid]) ),  detailsTable = Table.FromRecords(detailsList), withSchema = if schema <> null then SchemaTransformTable(detailsTable, schema) else detailsTable in withSchema; |

Steps in the Function GetVendorDetails:

1. Replace Placeholder - Replaces {vendor\_guid} in the provided url with the actual vendor\_guid.
2. **Fetch Data -** Makes a GET request to the details\_url with the default VRM request headers
3. Parse JSON -  Parses the response to extract the vendor's details
4. Add Vendor GUID - Adds the vendor\_guid to the record for easier identification.
5. Return Result

Steps in the Main Function BitSightSecurityVendorsDetails.Feed:

1. Retrieve All Vendors: Calls BitSightSecurityAllVendors.Feed to fetch a table of all vendors, excluding schema transformations.
2. Extract Relevant Columns: Extracts the vendor\_guid and type columns from the fetched data.
3. Fetch Vendor Details: For each vendor, calls GetVendorDetails based on its type:
	1. Managed vendors use managedVendorsDetailsURL.
	2. Monitored vendors use monitoredVendorsDetailsURL.
4. Convert to Table: Converts the list of detailed records into a table.
5. Apply Schema (Optional): If a schema is provided, applies schema transformations to the table.

Complexity: One call per Vendor (N Vendors → N Calls)

### Vendor Rating

Function 1: GetVendorRating: This function retrieves the rating details for a specific vendor by replacing the {bst\_entity\_guid} placeholder in the URL with the vendor's unique identifier and fetching data from the resulting API endpoint.

|  |
| --- |
| GetVendorRating = (url as text, bst\_entity\_guid as text) => let details\_url = Text.Replace(url, "{bst\_entity\_guid}", bst\_entity\_guid), source = Web.Contents(details\_url, [Headers = DefaultVRMRequestHeaders]), json = Json.Document(source), // Check if the response contains error-related fields detailsRecord = Record.AddField(json, "bst\_entity\_guid", bst\_entity\_guid) in detailsRecord; |

Steps:

1. **Replace Placeholder**: Replaces {bst\_entity\_guid} in the provided url with the actual bst\_entity\_guid.
2. **Fetch Data**: Makes a GET request to the constructed URL using default VRM headers.
3. **Parse JSON**: Parses the API response to extract rating details.
4. **Add Vendor GUID**: Adds the bst\_entity\_guid to the record for easier traceability.
5. **Return Result**: Returns the detailed rating record for the specified vendor.

Function 2: BitSightSecurityVendorRating.Feed:
This function fetches and processes vendor ratings for both "Managed" and "Monitored" vendors, ensuring that only vendors with valid bst\_entity\_guid are included.

|  |
| --- |
| BitSightSecurityVendorRating.Feed = (url as text, endpoint as text, flag as text, optional schema as table) => let // Get the managed and monitored vendors table allVendors = BitSightSecurityVendorsDetails.Feed(monitoredVendorsDetailsURL, endpoint, flag, null), // Check if managedVendors is empty isVendorsEmpty = Table.IsEmpty(allVendors),  // If managedVendors is not empty, get the vendor\_guids and fetch details detailsTable = if not isVendorsEmpty then let vendorGuids = Table.SelectColumns(allVendors, {"bst\_entity\_guid"}), nonNullGuids = Table.SelectRows(vendorGuids, each [bst\_entity\_guid] <> null), // Remove null values // Fetch details for each vendor\_guid and create a new table detailsList = List.Transform(nonNullGuids[bst\_entity\_guid], each GetVendorRating(url, \_)), detailsTable = Table.FromRecords(detailsList), #"Expanded ratings" = Table.ExpandListColumn(detailsTable, "ratings"), #"Expanded ratings1" = Table.ExpandRecordColumn(#"Expanded ratings", "ratings", {"rating\_date", "rating"}, {"ratings.rating\_date", "ratings.rating"}), finalTable = Table.RenameColumns(#"Expanded ratings1",{{"ratings.rating", "rating"}, {"ratings.rating\_date", "rating\_date"}}) in  finalTable else #table({}, {}),  withSchema = if schema <> null then SchemaTransformTable(detailsTable, schema) else detailsTable in withSchema; |

Steps:

1. **Retrieve All Vendors**: Calls BitSightSecurityVendorsDetails.Feed to get a table of all vendors (using monitoredVendorsDetailsURL).
2. **Check for Empty Data**: Verifies if the returned vendor table is empty.
3. **Filter Non-Empty Vendors**: If vendors exist, they select the bst\_entity\_guid column and filter out any rows with null values.
4. **Fetch Vendor Ratings**: Uses GetVendorRating to retrieve rating details for each bst\_entity\_guid and constructs a table from the resulting list of records.
5. **Expand Ratings**: Expands nested rating data into separate columns:
	* Expands the ratings list.
	* Extracts and renames the rating\_date and rating fields.
6. **Apply Schema (Optional)**: If a schema is provided, applies schema transformations to standardize the output.
7. **Return Result**: Returns the processed ratings table.

Complexity: One call per Vendor Details (N Vendors → 2N Vendor Ratings Calls)

### Vendor Info

Function 1: GetVendorInfo
This function fetches detailed information for a specific vendor by replacing the {vendor\_guid} placeholder in the URL with the vendor's unique identifier and retrieving data from the resulting API endpoint.

|  |
| --- |
| GetVendorInfo = (url as text, vendor\_guid as text) => let details\_url = Text.Replace(url, "{vendor\_guid}", vendor\_guid), source = Web.Contents(details\_url, [Headers = DefaultVRMRequestHeaders]), json = Json.Document(source) in json; |

Steps:

1. **Replace Placeholder**: Replaces {vendor\_guid} in the provided url with the actual vendor\_guid.
2. **Fetch Data**: Makes a GET request to the constructed URL using default VRM headers.
3. **Parse JSON**: Parses the API response and returns the vendor's information as a JSON object.

Function 2: BitSightSecurityVendorInfo.Feed
This function retrieves detailed information for all vendors and consolidates the data into a table, applying an optional schema transformation if provided.

|  |
| --- |
| BitSightSecurityVendorInfo.Feed = (url as text, endpoint as text, flag as text, optional schema as table) => let // Get the vendors table allVendors = BitSightSecurityAllVendors.Feed(allVendorsURL, endpoint, flag, null), // Check if allVendors is empty isVendorsEmpty = Table.IsEmpty(allVendors),  // If managedVendors is not empty, get the vendor\_guids and fetch details detailsTable = if not isVendorsEmpty then let vendorGuids = Table.SelectColumns(allVendors, {"vendor\_guid"}), // Fetch details for each vendor\_guid and create a new table detailsList = List.Transform(vendorGuids[vendor\_guid], each GetVendorInfo(url, \_)), detailsTable = Table.FromRecords(detailsList), finalTable = Table.RenameColumns(detailsTable,{{"guid", "vendor\_guid"}}) in  finalTable else #table({}, {}),  withSchema = if schema <> null then SchemaTransformTable(detailsTable, schema) else detailsTable in withSchema; |

Steps:

1. **Retrieve All Vendors**: Calls BitSightSecurityAllVendors.Feed to get a table of all vendors.
2. **Check for Empty Data**: Verifies if the returned vendor table is empty.
3. **Fetch Vendor Info (if not empty)**: If vendors exist:
	* Select the vendor\_guid column.
	* Calls GetVendorInfo for each vendor\_guid to fetch detailed vendor information.
	* Convert the list of records into a table.
	* Renames the guid column to vendor\_guid for consistency.
4. **Handle Empty Data**: Returns an empty table if no vendors are found.
5. **Apply Schema (Optional)**: If a schema is provided, applies schema transformations to align the data structure with predefined expectations.
6. **Return Result**: Outputs the final table with detailed vendor information.

Complexity: One call per Vendor Details (N Vendors → N Vendor Info Calls )

### Vendor Contacts

Function 1: GetVendorContact
This function retrieves contact information for a specific vendor by replacing the {vendor\_guid} placeholder in the URL with the vendor's unique identifier. It adds the vendor\_guid to each contact record for easier tracking.

|  |
| --- |
| GetVendorContact = (url as text, vendor\_guid as text) => let details\_url = Text.Replace(url, "{vendor\_guid}", vendor\_guid), source = Web.Contents(details\_url, [Headers = DefaultVRMRequestHeaders]), json = Json.Document(source), // Transform each contact record to add vendor\_guid to each contact detailsList = List.Transform(json, each Record.AddField(\_, "vendor\_guid", vendor\_guid)) in detailsList; |

Steps:

1. **Replace Placeholder**: Replaces {vendor\_guid} in the provided url with the actual vendor\_guid.
2. **Fetch Data**: Makes a GET request to the constructed details\_url using default VRM headers.
3. **Parse JSON**: Parses the API response to retrieve the list of contacts.
4. **Add Vendor GUID**: Adds the vendor\_guid to each contact record to maintain a link between the contact and the vendor.
5. **Return Result**: Outputs the transformed list of contact records.

Function 2: BitSightSecurityVendorContacts.Feed
This function retrieves and consolidates contact information for all vendors by fetching details for each vendor and combining them into a single table.

|  |
| --- |
| BitSightSecurityVendorContacts.Feed = (url as text, endpoint as text, flag as text, optional schema as table) => let // Get the managed and monitored vendors table allVendors = BitSightSecurityVendorsDetails.Feed(monitoredVendorsDetailsURL, endpoint, flag, null), isCombinedVendorsEmpty = Table.IsEmpty(allVendors), detailsTable = if not isCombinedVendorsEmpty then let vendorGuids = Table.SelectColumns(allVendors, {"vendor\_guid"}), // Fetch details for each vendor\_guid and create a new table detailsListOfLists = List.Transform(vendorGuids[vendor\_guid], each GetVendorContact(url, \_)), // Flatten the list of lists into a single list of records flattenedDetailsList = List.Combine(detailsListOfLists), detailsTable = Table.FromRecords(flattenedDetailsList) in  detailsTable else #table({}, {}), withSchema = if schema <> null then SchemaTransformTable(detailsTable, schema) else detailsTable in withSchema; |

Steps:

1. **Retrieve All Vendors**: Calls BitSightSecurityVendorsDetails.Feed to fetch details of all vendors (managed and monitored).
2. **Check for Empty Data**: Verifies if the combined vendor table is empty.
3. **Fetch Vendor Contacts (if not empty)**: If vendors exist:
	* Select the vendor\_guid column.
	* Calls GetVendorContact for each vendor\_guid to fetch contact details.
	* Combines the returned lists of contacts into a single flattened list.
	* Convert the flattened list into a table.
4. **Handle Empty Data**: Returns an empty table if no vendors are found.
5. **Apply Schema (Optional)**: If a schema is provided, applies schema transformations to align the data structure with predefined expectations.
6. **Return Result**: Outputs the final table with vendor contact information.

Complexity: One call per Vendor Details (N Vendors → N Vendor Contacts Calls )

### Vendor Internal Contacts

Same as Vendor Contacts (uses same auxiliar function (GetVendorContact). Just does it to a different endpoint

|  |
| --- |
| BitSightSecurityInternalContacts.Feed = (url as text, endpoint as text, flag as text, optional schema as table) => let // Get the managed and monitored vendors table allVendors = BitSightSecurityVendorsDetails.Feed(monitoredVendorsDetailsURL, endpoint, flag, null), isCombinedVendorsEmpty = Table.IsEmpty(allVendors), detailsTable = if not isCombinedVendorsEmpty then let vendorGuids = Table.SelectColumns(allVendors, {"vendor\_guid"}), // Fetch details for each vendor\_guid and create a new table detailsListOfLists = List.Transform(vendorGuids[vendor\_guid], each GetVendorContact(url, \_)), // Flatten the list of lists into a single list of records flattenedDetailsList = List.Combine(detailsListOfLists), detailsTable = Table.FromRecords(flattenedDetailsList) in  detailsTable else #table({}, {}), withSchema = if schema <> null then SchemaTransformTable(detailsTable, schema) else detailsTable in withSchema; |

Complexity: One call per Vendor Details (N Vendors → N Vendor Internal Contacts Calls )

### Vendor Findings

Function: BitSightSecurityFindings.Feed
This function retrieves and processes a list of security findings from a specified API endpoint. It converts the JSON response into a structured table, optionally applying a schema transformation.

|  |
| --- |
| BitSightSecurityFindings.Feed = (url as text, endpoint as text, flag as text, optional schema as table) => let source = Web.Contents(url, [ Headers = DefaultVRMRequestHeaders, Content = Json.FromValue([]) ]), json = Json.Document(source), resultsList = json[results], resultsTable = Table.FromList(resultsList, Splitter.SplitByNothing(), null, null, ExtraValues.Error), table\_fields = Record.FieldNames(Table.FirstValue(resultsTable, [Empty = null])), expanded = Table.ExpandRecordColumn(resultsTable, "Column1", table\_fields), withSchema = if schema <> null then SchemaTransformTable(expanded, schema) else expanded, output = withSchema in output; |

Steps:

1. **Fetch Data**: Makes a POST request to the provided url with DefaultVRMRequestHeaders and an empty JSON payload (Json.FromValue([])). Retrieves the API response as raw JSON.
2. **Parse JSON**: Converts the JSON response to a structured format using Json.Document. Extracts the results field, which contains a list of security findings.
3. **Create Findings Table**: Converts the results list into a table (Table.FromList), handling any extra values as errors.
4. **Extract Fields**: Identifies the fields in the first record of the table to determine the column names. Expand the nested records in the table using Table.ExpandRecordColumn, creating a flat table with all relevant fields.
5. **Apply Schema (Optional)**:
	* If a schema is provided, apply schema transformations to align the data with predefined requirements.
	* If no schema is provided, the expanded table is used as-is.
6. **Return Result**: Outputs the processed table, which contains the security findings in a structured format.

Complexity: One API call

### Findings Details

Function 1: GetFindingsDetails
This function retrieves detailed information about a specific security finding by replacing a placeholder in the URL with the finding\_guid.

|  |
| --- |
| GetFindingsDetails = (url as text, finding\_guid as text) => let details\_url = Text.Replace(url, "{finding\_guid}", finding\_guid), source = Web.Contents(details\_url, [Headers = DefaultVRMRequestHeaders]), json = Json.Document(source) in json; |

Steps:

1. **Replace Placeholder**: Replaces the {finding\_guid} in the provided url with the actual finding\_guid.
2. **Fetch Data**: Makes a GET request to the details\_url using DefaultVRMRequestHeaders.
3. **Parse JSON**: Converts the JSON response into a structured format and returns it.

Function 2: BitSightSecurityFindingsDetails.Feed
This function retrieves a list of security findings and fetches detailed information for each finding, returning the data as a structured table.

|  |
| --- |
| BitSightSecurityFindingsDetails.Feed = (url as text, endpoint as text, flag as text, optional schema as table) => let // Get the managed vendors table findingsVendors = BitSightSecurityFindings.Feed(allVendorFindingsURL, endpoint, flag, null),// Check if managedVendors is empty isfindingsVendorsEmpty = Table.IsEmpty(findingsVendors),  // If managedVendors is not empty, get the vendor\_guids and fetch details detailsTable = if not isfindingsVendorsEmpty then let vendorGuids = Table.SelectColumns(findingsVendors, {"guid"}), // Fetch details for each vendor\_guid and create a new table detailsList = List.Transform(vendorGuids[guid], each GetFindingsDetails(url, \_)), detailsTable = Table.FromRecords(detailsList) in  detailsTable else #table({}, {}),  withSchema = if schema <> null then SchemaTransformTable(detailsTable, schema) else detailsTable in withSchema; |

Steps:

1. **Retrieve Findings**: Calls BitSightSecurityFindings.Feed to fetch a table of all security findings from the allVendorFindingsURL.
2. **Check for Empty Results**: Verifies whether the findings table is empty using Table.IsEmpty.
3. **Fetch Finding Details**:
	* If the findings table is not empty:
		+ Extracts the guid column, which contains unique identifiers for the findings.
		+ Iterates over each guid using List.Transform to call GetFindingsDetails and retrieve detailed information for each finding.
		+ Convert the list of detailed records into a structured table (Table.FromRecords).
4. **Apply Schema (Optional)**: If a schema is provided, applies schema transformations to the table. If no schema is provided, the raw table is returned.
5. **Return Result**: Outputs the processed findings detail table.

Complexity: One call per Finding (N Findings → N Findings Details Calls )

### Internal Questionnaires:

Function: BitSightSecurityInternalQuestionnaires.Feed

This function retrieves internal questionnaire data for vendors and processes it into a structured table.

|  |
| --- |
| BitSightSecurityInternalQuestionnaires.Feed = (url as text, endpoint as text, flag as text, optional schema as table) => let // Get the vendors table allVendors = BitSightSecurityAllVendors.Feed(allVendorsURL, endpoint, flag, null), // Check if allVendors is empty isVendorsEmpty = Table.IsEmpty(allVendors),  // If managedVendors is not empty, get the vendor\_guids and fetch details detailsTable = if not isVendorsEmpty then let vendorGuids = Table.SelectColumns(allVendors, {"vendor\_guid"}), // Fetch details for each vendor\_guid and create a new table detailsList = List.Transform(vendorGuids[vendor\_guid], each GetVendorQuestion(url, \_)), detailsTable = Table.Combine(detailsList), //#"Removed Columns" = Table.RemoveColumns(detailsTable,{"questionnaire\_guid", "logo\_url"}) finalTable = Table.RenameColumns(detailsTable,{{"guid", "questionnaire\_guid"}}) //#"Removed Duplicates" = Table.Distinct(detailsTable, {"guid"}) in  finalTable else #table({}, {}),  withSchema = if schema <> null then SchemaTransformTable(detailsTable, schema) else detailsTable in withSchema; |

Steps:

1. **Retrieve All Vendors**: Calls BitSightSecurityAllVendors.Feed to fetch a table of all vendors from the allVendorsURL.
2. **Check for Empty Results**: Uses Table.IsEmpty to check if the allVendors table contains any data.
3. **Fetch Questionnaire Details**:
	* If the allVendors table is not empty:
		+ Extracts the vendor\_guid column, which identifies each vendor.
		+ Iterates over the list of vendor\_guids using List.Transform to call GetVendorQuestion for each vendor and retrieve their questionnaire data.
		+ Combines the resulting lists of questionnaire data into a single table using Table.Combine.
		+ Renames the guid column to questionnaire\_guid.
4. **Apply Schema (Optional)**: If a schema is provided, applies schema transformations to ensure consistent data structure. If no schema is provided, return the raw table.
5. **Return Result**: Outputs the processed table containing internal questionnaire data for all vendors.

Complexity: One call per Vendor (N Vendors → N Questionnaires Calls )

### Internal Questionnaires Categories

Function 1: GetQuestionCat
This function retrieves category details for a specific questionnaire by replacing the {questionnaire\_guid} placeholder in the URL with the provided questionnaire\_guid. It then fetches and parses the category information.

|  |
| --- |
| GetQuestionCat = (url as text, questionnaire\_guid as text) => let details\_url = Text.Replace(url, "{questionnaire\_guid}", questionnaire\_guid), source = Web.Contents(details\_url, [Headers = DefaultVRMRequestHeaders]), json = Json.Document(source) in json; |

Steps**:**

1. **Replace Placeholder:** Replaces {questionnaire\_guid} in the provided URL with the actual questionnaire\_guid.
2. **Fetch Data:** Makes a GET request to the constructed details\_url using default VRM headers.
3. **Parse JSON:** Parses the API response to retrieve the category details.
4. **Return Result:** Outputs the parsed JSON object containing the category information.

**Function 2: BitSightSecurityInternalQuestionnairesCategories.Feed**
This function retrieves internal questionnaire categories by first fetching the internal questionnaires table, checking if it’s empty, and then fetching the category details for each unique questionnaire\_guid. If data is available, the function returns a structured table with the categories.

|  |
| --- |
| BitSightSecurityInternalQuestionnairesCategories.Feed = (url as text, endpoint as text, flag as text, optional schema as table) => let // Get the internal quest table allInternalQ = BitSightSecurityInternalQuestionnaires.Feed(internalQuestionnairesURL, endpoint, flag, null), // Check if table is empty isEmpty = Table.IsEmpty(allInternalQ),  // If isEmpty is not empty, get the questionnaire\_guid and fetch categories detailsTable = if not isEmpty then let removed\_dup = Table.Distinct(allInternalQ, {"questionnaire\_guid"}), table\_Guids = Table.SelectColumns(removed\_dup, {"questionnaire\_guid"}), // Fetch details for each vendor\_guid and questionnaire\_guid pair detailsList = List.Transform( Table.ToRecords(table\_Guids),  each GetQuestionCat(url, \_[questionnaire\_guid]) ), // Combine all the resulting tables into one combinedDetailsTable = Table.FromRecords(detailsList), #"Removed Other Columns" = Table.SelectColumns(combinedDetailsTable,{"guid", "categories"}), #"Expanded categories" = Table.ExpandListColumn(#"Removed Other Columns", "categories"), #"Expanded categories1" = Table.ExpandRecordColumn(#"Expanded categories", "categories", {"guid", "name"}, {"categories.guid", "categories.name"}), #"Renamed Columns" = Table.RenameColumns(#"Expanded categories1",{{"guid", "questionnaire\_guid"}, {"categories.guid", "category\_guid"}, {"categories.name", "name"}}) in  #"Renamed Columns" else #table({}, {}),  withSchema = if schema <> null then SchemaTransformTable(detailsTable, schema) else detailsTable in withSchema; |

Steps**:**

1. **Retrieve All Internal Questionnaires:** Calls BitSightSecurityInternalQuestionnaires.Feed to fetch details of internal questionnaires.
2. **Check for Empty Data:** Verifies if the fetched data (allInternalQ) is empty.
3. **Fetch Category Details (if not empty):** If the internal questionnaires table contains data:
	* Removes duplicates and selects the questionnaire\_guid column.
	* Calls GetQuestionCat for each unique questionnaire\_guid to fetch category details.
	* Combine all the fetched details into a single table and renames columns accordingly.
4. **Handle Empty Data:** Returns an empty table if no internal questionnaires are found.
5. **Apply Schema (Optional):** If a schema is provided, applies schema transformations to align the data structure with predefined expectations.
6. **Return Result:** Outputs the final table with internal questionnaire category details.

Complexity: One call per Questionnaire (N Vendors → N Questionnaires Calls → Y Questionnaires → Y Questionnaire Categories Calls )

### Internal Questionnaires Questions

**Function 1: GetCategoryQuestion**
This function retrieves a list of questions for a specific vendor and category by replacing the {vendor\_guid} and {category\_guid} placeholders in the URL with the respective values. It then fetches the data and parses the results into a table.

|  |
| --- |
|  GetCategoryQuestion = (url as text, vendor\_guid as text, category\_guid as text) => let // Replace vendor\_guid and category\_guid in the URL details\_url = Text.Replace(Text.Replace(url, "{vendor\_guid}", vendor\_guid), "{category\_guid}", category\_guid), // Fetch data from the constructed URL source = Web.Contents(details\_url, [Headers = DefaultVRMRequestHeaders]), // Parse JSON response json = Json.Document(source), resultsTable = Table.FromList(json, Splitter.SplitByNothing(), null, null, ExtraValues.Error) in resultsTable; |

**Steps:**

1. **Replace Placeholders:** Replaces {vendor\_guid} and {category\_guid} in the provided URL with the actual values.
2. **Fetch Data:** Makes a GET request to the constructed details\_url using default VRM headers.
3. **Parse JSON:** Parses the API response to retrieve the list of questions.
4. **Return Result:** Converts the list of questions into a table and returns it.

**Function 2: BitSightSecurityQuestCategoriesInternalAuxiliar**
This function retrieves vendor-category pair details by first fetching the internal questionnaires, then fetching category details for each vendor and questionnaire. It returns a table with the relevant vendor and category information.

|  |
| --- |
| BitSightSecurityQuestCategoriesInternalAuxiliar = (url as text, endpoint as text, flag as text, optional schema as table) => let // Get the quest table allInternalQ = BitSightSecurityInternalQuestionnaires.Feed(internalQuestionnairesURL, endpoint, flag, null), // Check if allVendors is empty isEmpty = Table.IsEmpty(allInternalQ),  // If managedVendors is not empty, get the vendor\_guids and fetch details detailsTable = if not isEmpty then let table\_Guids = Table.SelectColumns(allInternalQ, {"questionnaire\_guid", "vendor\_guid"}), // Fetch details for each vendor\_guid and questionnaire\_guid pair detailsList = List.Transform( Table.ToRecords(table\_Guids),  each GetQuestionCatWithVendorID(url, \_[questionnaire\_guid], \_[vendor\_guid]) ), // Combine all the resulting tables into one combinedDetailsTable = Table.FromRecords(detailsList), #"Removed Other Columns" = Table.SelectColumns(combinedDetailsTable,{"categories", "vendor\_guid"}), #"Expanded categories" = Table.ExpandListColumn(#"Removed Other Columns", "categories"), #"Expanded categories1" = Table.ExpandRecordColumn(#"Expanded categories", "categories", {"guid"}, {"categories.guid"}), #"Renamed Columns" = Table.RenameColumns(#"Expanded categories1",{{"categories.guid", "category\_guid"}}) in  #"Renamed Columns" else #table({}, {}), withSchema = detailsTable in withSchema; |

Steps**:**

1. **Retrieve All Internal Questionnaires:** Calls BitSightSecurityInternalQuestionnaires.Feed to fetch details of internal questionnaires.
2. **Check for Empty Data:** Verifies if the internal questionnaires data (allInternalQ) is empty.
3. **Fetch Vendor-Category Pairs (if not empty):** If data is available:
	* Selects the questionnaire\_guid and vendor\_guid columns from the data.
	* Calls GetQuestionCatWithVendorID for each vendor-category pair to fetch category details.
	* Combines the results into a single table and renames columns accordingly.
4. **Handle Empty Data:** Returns an empty table if no vendor-category pairs are found.
5. **Return Result:** Outputs the final table with vendor-category pair details.

**Function 3: GetQuestionCatWithVendorID**

This function retrieves category details for a specific questionnaire by replacing the {questionnaire\_guid} placeholder in the URL with the provided questionnaire\_guid. It also adds the vendor\_guid to the returned data for better tracking.

|  |
| --- |
| GetQuestionCatWithVendorID = (url as text, questionnaire\_guid as text, vendor\_guid as text) => let details\_url = Text.Replace(url, "{questionnaire\_guid}", questionnaire\_guid), source = Web.Contents(details\_url, [Headers = DefaultVRMRequestHeaders]), json = Json.Document(source), // Add questionnaire\_guid to each line detailsRecord = Record.AddField(json, "vendor\_guid", vendor\_guid) in detailsRecord; |

**Steps:**

1. **Replace Placeholder:** Replaces {questionnaire\_guid} in the provided URL with the actual questionnaire\_guid.
2. **Fetch Data:** Makes a GET request to the constructed details\_url using default VRM headers.
3. **Parse JSON:** Parses the API response to retrieve the category details.
4. **Add Vendor GUID:** Adds the vendor\_guid field to the resulting record to link it with the specific vendor.
5. **Return Result:** Outputs the modified record containing the category details along with the vendor\_guid.

**Function 4: BitSightSecurityInternalQuestionnairesQuestions.Feed**
This is the main function that retrieves questionnaire questions for each vendor-category pair by first fetching the vendor-category details, then calling GetCategoryQuestion to fetch the questions for each vendor-category. It returns a table with the question details.

|  |
| --- |
| BitSightSecurityInternalQuestionnairesQuestions.Feed = (url as text, endpoint as text, flag as text, optional schema as table) => let // Get the vendors and categories table vendorsCategories = BitSightSecurityQuestCategoriesInternalAuxiliar(questionnairesCategoriesURL, endpoint, flag, null),  // Check if the vendorsCategories table is empty isEmpty = Table.IsEmpty(vendorsCategories),  // Process the table if not empty detailsTable = if not isEmpty then let // Transform each row to include the results from GetCategoryQuestion detailsList = List.Transform( Table.ToRecords(vendorsCategories),  each  Record.AddField(\_, "question\_results",  GetCategoryQuestion(url, [vendor\_guid], [category\_guid]) ) ), // Convert the list of records into a table resultTable = Table.FromRecords(detailsList), #"Expanded question\_results" = Table.ExpandTableColumn(resultTable, "question\_results", {"Column1"}, {"question\_results.Column1"}), #"Expanded question\_results.Column1" = Table.ExpandRecordColumn(#"Expanded question\_results", "question\_results.Column1", {"guid", "question"}, {"question\_results.Column1.guid", "question\_results.Column1.question"}), finalTable = Table.RenameColumns(#"Expanded question\_results.Column1",{{"question\_results.Column1.guid", "question\_guid"}, {"question\_results.Column1.question", "question"}}) in finalTable else #table({}, {}), withSchema = if schema <> null then SchemaTransformTable(detailsTable, schema) else detailsTable in // Apply schema transformation if schema is provided withSchema; |

**Steps:**

1. **Retrieve Vendor-Category Pairs:** Calls BitSightSecurityQuestCategoriesInternalAuxiliar to fetch details of vendor-category pairs.
2. **Check for Empty Data:** Verifies if the fetched vendor-category data (vendorsCategories) is empty.
3. **Fetch Questionnaire Questions (if not empty):** If vendor-category pairs are available:
	* For each vendor-category pair, calls GetCategoryQuestion to fetch the questions.
	* Expands the question\_results column to extract the question details.
	* Renames columns to provide a clearer structure for the question data.
4. **Handle Empty Data:** Returns an empty table if no vendor-category pairs are found.
5. **Apply Schema (Optional):** If a schema is provided, applies schema transformations to align the data structure with predefined expectations.
6. **Return Result:** Outputs the final table with questionnaire question details.

Complexity: Get All Vendors (1 Call) → N Vendors → Get all Questionnaires (N Calls) → Y Questionnaires → For each pair Vendor/Questionnaire get Category → For each pair Vendor/Category get Question

### Internal Questionnaires Answers

The thought process of this is the same as the Internal Questionnaires Questions.

|  |
| --- |
| GetCategoryQuestion = (url as text, vendor\_guid as text, category\_guid as text) => let // Replace vendor\_guid and category\_guid in the URL details\_url = Text.Replace(Text.Replace(url, "{vendor\_guid}", vendor\_guid), "{category\_guid}", category\_guid), // Fetch data from the constructed URL source = Web.Contents(details\_url, [Headers = DefaultVRMRequestHeaders]), // Parse JSON response json = Json.Document(source), resultsTable = Table.FromList(json, Splitter.SplitByNothing(), null, null, ExtraValues.Error) in resultsTable; |

|  |
| --- |
| BitSightSecurityQuestCategoriesAuxiliar = (url as text, endpoint as text, flag as text, optional schema as table) => let // Get the quest table allInternalQ = BitSightSecurityExternalQuestionnaires.Feed(externalQuestionnairesURL, endpoint, flag, null), // Check if allVendors is empty isEmpty = Table.IsEmpty(allInternalQ),  // If managedVendors is not empty, get the vendor\_guids and fetch details detailsTable = if not isEmpty then let table\_Guids = Table.SelectColumns(allInternalQ, {"questionnaire\_guid", "vendor\_guid"}), // Fetch details for each vendor\_guid and questionnaire\_guid pair detailsList = List.Transform( Table.ToRecords(table\_Guids),  each GetQuestionCatWithVendorID(url, \_[questionnaire\_guid], \_[vendor\_guid]) ), // Combine all the resulting tables into one combinedDetailsTable = Table.FromRecords(detailsList), #"Removed Other Columns" = Table.SelectColumns(combinedDetailsTable,{"categories", "vendor\_guid"}), #"Expanded categories" = Table.ExpandListColumn(#"Removed Other Columns", "categories"), #"Expanded categories1" = Table.ExpandRecordColumn(#"Expanded categories", "categories", {"guid"}, {"categories.guid"}), #"Renamed Columns" = Table.RenameColumns(#"Expanded categories1",{{"categories.guid", "category\_guid"}}) in  #"Renamed Columns" else #table({}, {}), withSchema = detailsTable in withSchema; |

|  |
| --- |
| GetQuestionCatWithVendorID = (url as text, questionnaire\_guid as text, vendor\_guid as text) => let details\_url = Text.Replace(url, "{questionnaire\_guid}", questionnaire\_guid), source = Web.Contents(details\_url, [Headers = DefaultVRMRequestHeaders]), json = Json.Document(source), // Add questionnaire\_guid to each line detailsRecord = Record.AddField(json, "vendor\_guid", vendor\_guid) in detailsRecord; |

|  |
| --- |
| BitSightSecurityInternalQuestionnairesQuestions.Feed = (url as text, endpoint as text, flag as text, optional schema as table) => let // Get the vendors and categories table vendorsCategories = BitSightSecurityQuestCategoriesInternalAuxiliar(questionnairesCategoriesURL, endpoint, flag, null),  // Check if the vendorsCategories table is empty isEmpty = Table.IsEmpty(vendorsCategories),  // Process the table if not empty detailsTable = if not isEmpty then let // Transform each row to include the results from GetCategoryQuestion detailsList = List.Transform( Table.ToRecords(vendorsCategories),  each  Record.AddField(\_, "question\_results",  GetCategoryQuestion(url, [vendor\_guid], [category\_guid]) ) ), // Convert the list of records into a table resultTable = Table.FromRecords(detailsList), #"Expanded question\_results" = Table.ExpandTableColumn(resultTable, "question\_results", {"Column1"}, {"question\_results.Column1"}), #"Expanded question\_results.Column1" = Table.ExpandRecordColumn(#"Expanded question\_results", "question\_results.Column1", {"guid", "question"}, {"question\_results.Column1.guid", "question\_results.Column1.question"}), finalTable = Table.RenameColumns(#"Expanded question\_results.Column1",{{"question\_results.Column1.guid", "question\_guid"}, {"question\_results.Column1.question", "question"}}) in finalTable else #table({}, {}), withSchema = if schema <> null then SchemaTransformTable(detailsTable, schema) else detailsTable in // Apply schema transformation if schema is provided withSchema; |

### Internal Questionnaires Progress

**Function 1: GetQuestionProgress**

This function retrieves the progress of a specific questionnaire for a vendor by replacing the {vendor\_guid} and {questionnaire\_guid} placeholders in the URL with the provided values. It then fetches the data and appends the vendor\_guid and questionnaire\_guid to the result.

|  |
| --- |
| GetQuestionProgress = (url as text, vendor\_guid as text, questionnaire\_guid as text) => let // Replace vendor\_guid and questionnaire\_guid in the URL details\_url = Text.Replace(Text.Replace(url, "{vendor\_guid}", vendor\_guid), "{questionnaire\_guid}", questionnaire\_guid), // Fetch data from the constructed URL source = Web.Contents(details\_url, [Headers = DefaultVRMRequestHeaders]), // Parse JSON response json = Json.Document(source), // Add questionnaire\_guid to each line detailsRecord = Record.AddField(json, "questionnaire\_guid", questionnaire\_guid), detailsRecord2 = Record.AddField(detailsRecord, "vendor\_guid", vendor\_guid) in detailsRecord2; |

**Steps:**

1. **Replace Placeholders:** Replaces {vendor\_guid} and {questionnaire\_guid} in the URL with the respective vendor\_guid and questionnaire\_guid.
2. **Fetch Data:** Makes a GET request to the constructed details\_url using default VRM headers.
3. **Parse JSON:** Parses the API response to retrieve the questionnaire progress data.
4. **Add Questionnaire GUID:** Adds the questionnaire\_guid to the resulting record.
5. **Add Vendor GUID:** Adds the vendor\_guid to the resulting record to maintain the link between the vendor and the questionnaire.
6. **Return Result:** Outputs the modified record containing the progress data with both questionnaire\_guid and vendor\_guid.

**Function 2: BitSightSecurityInternalQuestionnairesProgress.Feed**
This function retrieves and consolidates the progress data for each questionnaire and vendor. It first fetches the internal questionnaires, checks if they contain any data, and then calls GetQuestionProgress for each vendor-questionnaire pair to gather the progress details.

|  |
| --- |
| BitSightSecurityInternalQuestionnairesProgress.Feed = (url as text, endpoint as text, flag as text, optional schema as table) => let // Get the allInternalQ table allInternalQ = BitSightSecurityInternalQuestionnaires.Feed(internalQuestionnairesURL, endpoint, flag, null), // Check if allInternalQ is empty isEmpty = Table.IsEmpty(allInternalQ),  // If managedVendors is not empty, get the vendor\_guids and fetch details detailsTable = if not isEmpty then let table\_Guids = Table.SelectColumns(allInternalQ, {"questionnaire\_guid", "vendor\_guid"}), // Fetch details for each vendor\_guid and questionnaire\_guid pair detailsList = List.Transform( Table.ToRecords(table\_Guids),  each GetQuestionProgress(url, \_[vendor\_guid], \_[questionnaire\_guid]) ), // Combine all the resulting tables into one combinedDetailsTable = Table.FromRecords(detailsList) in  combinedDetailsTable else #table({}, {}),  withSchema = if schema <> null then SchemaTransformTable(detailsTable, schema) else detailsTable in withSchema; |

**Steps:**

1. **Retrieve All Internal Questionnaires:** Calls BitSightSecurityInternalQuestionnaires.Feed to fetch details of internal questionnaires.
2. **Check for Empty Data:** Verifies if the internal questionnaires data (allInternalQ) is empty.
3. **Fetch Questionnaire Progress (if not empty):** If internal questionnaires data is available:
	* Selects the questionnaire\_guid and vendor\_guid columns from the data.
	* Calls GetQuestionProgress for each vendor-questionnaire pair to fetch progress details.
	* Combines the results into a single table.
4. **Handle Empty Data:** Returns an empty table if no questionnaires are found.
5. **Apply Schema (Optional):** If a schema is provided, applies schema transformations to align the data structure with predefined expectations.
6. **Return Result:** Outputs the final table with questionnaire progress data.

Complexity: Get all Vendors → N Vendors → Get all Questionnaires (One call per Vendor) → For each pair Vendor/Questionnaire get the progress

### External Questionnaires

Follows the same thought process and structure of the internal questionnaires section.

### External Questionnaires Categories

Follows the same thought process and structure of the internal questionnaires section.

### External Questionnaires Answers

Follows the same thought process and structure of the internal questionnaires section.

### External Questionnaires Questions

Follows the same thought process and structure of the internal questionnaires section.

### External Questionnaires Progress

Follows the same thought process and structure of the internal questionnaires section.

### Vendor Requirements

**Function: BitSightSecurityRequirements.Feed**
This function retrieves and processes requirements data for vendors. It fetches the list of all vendors, then makes paginated API requests (in batches of 50 vendor GUIDs) to retrieve detailed requirements data. The results are parsed, expanded, and transformed into a structured table.

|  |
| --- |
| BitSightSecurityRequirements.Feed = (url as text, endpoint as text, flag as text, optional schema as table) => let // Get the vendors table allVendors = BitSightSecurityAllVendors.Feed(allVendorsURL, endpoint, flag, null), // Check if allVendors is empty isVendorsEmpty = Table.IsEmpty(allVendors),  // If allVendors is not empty, process the requirements requirementsTable = if not isVendorsEmpty then let // Extract vendor\_guids as a list vendorGuidsList = Table.Column(allVendors, "vendor\_guid"), // Function to split list into chunks of 50 ChunkList = List.Split(vendorGuidsList, 50), // Function to fetch data for a given chunk FetchDataForChunk = (chunk as list) => let vendorGuidsQuery = Text.Combine(List.Transform(chunk, each "vendor\_guids=" & \_), "&"), details\_url = url & "?" & vendorGuidsQuery, source = Web.Contents(details\_url, [Headers = DefaultVRMRequestHeaders]), json = Json.Document(source), requirementsTable = Table.FromList(json, Splitter.SplitByNothing(), null, null, ExtraValues.Error), isrequirementsTableEmpty = Table.IsEmpty(requirementsTable), requirementsTableaux = if not isrequirementsTableEmpty then let #"Expanded Column1" = Table.ExpandRecordColumn(requirementsTable, "Column1", {"requirements\_fulfilled", "progress\_average", "total\_documents\_count", "total\_completed\_documents\_count", "vendor\_guid", "progress"}, {"Column1.requirements\_fulfilled", "Column1.progress\_average", "Column1.total\_documents\_count", "Column1.total\_completed\_documents\_count", "Column1.vendor\_guid", "Column1.progress"}), #"Renamed Columns" = Table.RenameColumns(#"Expanded Column1", {{"Column1.requirements\_fulfilled", "requirements\_fulfilled"}, {"Column1.progress\_average", "progress\_average"}, {"Column1.total\_documents\_count", "total\_documents\_count"}, {"Column1.total\_completed\_documents\_count", "total\_completed\_documents\_count"}, {"Column1.vendor\_guid", "vendor\_guid"}, {"Column1.progress", "progress"}}), expanded = Table.ExpandListColumn(#"Renamed Columns", "progress"), #"Expanded progress" = Table.ExpandRecordColumn(expanded, "progress", {"total\_requirement\_completion", "number\_of\_requirements", "requirement\_guid", "requirement\_name", "is\_completed"}, {"progress.total\_requirement\_completion", "progress.number\_of\_requirements", "progress.requirement\_guid", "progress.requirement\_name", "progress.is\_completed"}), #"Renamed Columns2" = Table.RenameColumns(#"Expanded progress", {{"progress.total\_requirement\_completion", "total\_requirement\_completion"}, {"progress.number\_of\_requirements", "number\_of\_requirements"}, {"progress.requirement\_guid", "requirement\_guid"}, {"progress.requirement\_name", "requirement\_name"}, {"progress.is\_completed", "is\_completed"}}) in #"Renamed Columns2" else #table({}, {}) in requirementsTableaux,  // Fetch data for all chunks and combine them Results = List.Transform(ChunkList, each FetchDataForChunk(\_)), CombinedTable = Table.Combine(Results) in CombinedTable else #table({}, {}),  // Apply schema transformation if provided withSchema = if schema <> null then SchemaTransformTable(requirementsTable, schema) else requirementsTable in withSchema; |

**Steps:**

1. **Retrieve All Vendors:** Calls BitSightSecurityAllVendors.Feed to fetch details of all vendors.
2. **Check for Empty Vendor Data:** Verifies if the vendor data (allVendors) is empty.
3. **Chunk Vendor GUIDs:** Extracts the vendor\_guid column and splits it into chunks of 50 GUIDs each to prevent issues with URL length limits on the API.
4. **Process Each Chunk:** For every chunk of 50 vendor GUIDs:
	1. Constructs a query string (e.g., vendor\_guids=guid1&vendor\_guids=guid2&...)
	2. Appends the query string to the base URL
	3. Makes a **GET** request to the constructed details\_url using default VRM headers
	4. Parses the JSON response and converts it into a table
5. **Combine Results:** All tables resulting from chunked calls are concatenated using Table.Combine.
6. **Handle Empty Data:** Returns an empty table if no vendor data is available.
7. **Apply Schema (Optional):** If a schema is provided, applies schema transformations to align the data structure with predefined expectations.
8. **Return Result:** Outputs the final table with requirements data.

**Complexity:**

* API Calls:
	+ Get All Vendors (1 call)
	+ Get Requirements (N calls, one per 50 vendors)
* Pagination:
	+ Automatic and internal to the function via List.Split on vendor GUIDs.

### Vendor Requirements Details

**Function:** GetRequirementDetails
This function retrieves detailed information for a specific requirement of a given vendor. It fetches the details using the requirement\_guid and vendor\_guid from a specified URL.

|  |
| --- |
| GetRequirementDetails = (url as text, vendor\_guid as text, requirement\_guid as text) => let // Replace vendor\_guid and questionnaire\_guid in the URL details\_url = Text.Replace(Text.Replace(url, "{vendor\_guid}", vendor\_guid), "{requirement\_guid}", requirement\_guid), // Fetch data from the constructed URL source = Web.Contents(details\_url, [Headers = DefaultVRMRequestHeaders]), // Parse JSON response json = Json.Document(source) in json; |

**Steps:**

1. **Replace placeholders in the URL:** Replaces {vendor\_guid} and {requirement\_guid} in the URL with the actual values.
2. **Fetch Data:** Makes a request to the constructed details\_url using the default VRM request headers.
3. **Parse JSON Response:** Converts the response JSON into a structured format for further processing.
4. **Return Data:** Outputs the parsed JSON as the result.

**Function:** BitSightSecurityRequirementsDetails.Feed
This function processes and retrieves detailed requirement data for vendors. It fetches the list of vendor requirements, extracts the necessary GUIDs, and fetches the corresponding detailed information using the GetRequirementDetails function.

|  |
| --- |
| BitSightSecurityRequirementsDetails.Feed = (url as text, endpoint as text, flag as text, optional schema as table) => let // Get the vendors table allReqs = BitSightSecurityRequirements.Feed(requirementsURL, endpoint, flag, null), // Check if allVendors is empty isReqsEmpty = Table.IsEmpty(allReqs),  // If allVendors is not empty, process the requirements requirementsDetailsTable = if not isReqsEmpty then let removed\_dup = Table.Distinct(allReqs, {"requirement\_guid", "vendor\_guid"}), table\_Guids = Table.SelectColumns(removed\_dup, {"requirement\_guid", "vendor\_guid"}),  detailsList = Table.AddColumn( table\_Guids,  "Details",  each GetRequirementDetails(url, [vendor\_guid], [requirement\_guid]) ), #"Expanded Details" = Table.ExpandRecordColumn(detailsList, "Details", {"name", "packages"}, {"Details.name", "Details.packages"}), #"Expanded Details.packages" = Table.ExpandListColumn(#"Expanded Details", "Details.packages"), #"Expanded Details.packages1" = Table.ExpandRecordColumn(#"Expanded Details.packages", "Details.packages", {"questionnaires\_segment", "certifications\_segment", "audits\_segment", "insurances\_segment"}, {"Details.packages.questionnaires\_segment", "Details.packages.certifications\_segment", "Details.packages.audits\_segment", "Details.packages.insurances\_segment"}), #"Expanded Details.packages.questionnaires\_segment" = Table.ExpandRecordColumn(#"Expanded Details.packages1", "Details.packages.questionnaires\_segment", {"artifacts"}, {"Details.packages.questionnaires\_segment.artifacts"}), #"Expanded Details.packages.questionnaires\_segment.artifacts" = Table.ExpandListColumn(#"Expanded Details.packages.questionnaires\_segment", "Details.packages.questionnaires\_segment.artifacts"), #"Expanded Details.packages.questionnaires\_segment.artifacts1" = Table.ExpandRecordColumn(#"Expanded Details.packages.questionnaires\_segment.artifacts", "Details.packages.questionnaires\_segment.artifacts", {"name", "score", "progress"}, {"Details.packages.questionnaires\_segment.artifacts.name", "Details.packages.questionnaires\_segment.artifacts.score", "Details.packages.questionnaires\_segment.artifacts.progress"}), #"Renamed Columns" = Table.RenameColumns(#"Expanded Details.packages.questionnaires\_segment.artifacts1",{{"Details.packages.questionnaires\_segment.artifacts.name", "requirements.questionnaires.name"}, {"Details.packages.questionnaires\_segment.artifacts.score", "requirements.questionnaires.score"}, {"Details.packages.questionnaires\_segment.artifacts.progress", "requirements.questionnaires.progress"}}), #"Expanded Details.packages.certifications\_segment" = Table.ExpandRecordColumn(#"Renamed Columns", "Details.packages.certifications\_segment", {"artifacts"}, {"Details.packages.certifications\_segment.artifacts"}), #"Expanded Details.packages.certifications\_segment.artifacts" = Table.ExpandListColumn(#"Expanded Details.packages.certifications\_segment", "Details.packages.certifications\_segment.artifacts"), #"Expanded Details.packages.certifications\_segment.artifacts1" = Table.ExpandRecordColumn(#"Expanded Details.packages.certifications\_segment.artifacts", "Details.packages.certifications\_segment.artifacts", {"name", "status"}, {"Details.packages.certifications\_segment.artifacts.name", "Details.packages.certifications\_segment.artifacts.status"}), #"Renamed Columns1" = Table.RenameColumns(#"Expanded Details.packages.certifications\_segment.artifacts1",{{"Details.packages.certifications\_segment.artifacts.name", "requirements.certifications.name"}, {"Details.packages.certifications\_segment.artifacts.status", "requirements.certifications.status"}}), #"Expanded Details.packages.audits\_segment" = Table.ExpandRecordColumn(#"Renamed Columns1", "Details.packages.audits\_segment", {"artifacts"}, {"Details.packages.audits\_segment.artifacts"}), #"Expanded Details.packages.audits\_segment.artifacts" = Table.ExpandListColumn(#"Expanded Details.packages.audits\_segment", "Details.packages.audits\_segment.artifacts"), #"Expanded Details.packages.audits\_segment.artifacts1" = Table.ExpandRecordColumn(#"Expanded Details.packages.audits\_segment.artifacts", "Details.packages.audits\_segment.artifacts", {"name", "status"}, {"Details.packages.audits\_segment.artifacts.name", "Details.packages.audits\_segment.artifacts.status"}), #"Renamed Columns2" = Table.RenameColumns(#"Expanded Details.packages.audits\_segment.artifacts1",{{"Details.packages.audits\_segment.artifacts.name", "requirements.audits.name"}, {"Details.packages.audits\_segment.artifacts.status", "requirements.audits.status"}}), #"Expanded Details.packages.insurances\_segment" = Table.ExpandRecordColumn(#"Renamed Columns2", "Details.packages.insurances\_segment", {"artifacts"}, {"Details.packages.insurances\_segment.artifacts"}), #"Expanded Details.packages.insurances\_segment.artifacts" = Table.ExpandListColumn(#"Expanded Details.packages.insurances\_segment", "Details.packages.insurances\_segment.artifacts"), #"Expanded Details.packages.insurances\_segment.artifacts1" = Table.ExpandRecordColumn(#"Expanded Details.packages.insurances\_segment.artifacts", "Details.packages.insurances\_segment.artifacts", {"insurances", "status"}, {"Details.packages.insurances\_segment.artifacts.insurances", "Details.packages.insurances\_segment.artifacts.status"}), #"Expanded Details.packages.insurances\_segment.artifacts.insurances" = Table.ExpandListColumn(#"Expanded Details.packages.insurances\_segment.artifacts1", "Details.packages.insurances\_segment.artifacts.insurances"), #"Renamed Columns3" = Table.RenameColumns(#"Expanded Details.packages.insurances\_segment.artifacts.insurances",{{"Details.packages.insurances\_segment.artifacts.insurances", "requirements.insurances.name"}, {"Details.packages.insurances\_segment.artifacts.status", "requirements.insurances.status"}}) in #"Renamed Columns3" else // Return an empty table if no vendors are available #table({}, {}),  // Apply schema transformation if provided withSchema = requirementsDetailsTable in withSchema; |

**Steps:**

1. **Retrieve Requirements Data:** Calls BitSightSecurityRequirements.Feed to get the list of all requirements data for vendors.
2. **Check for Empty Data:** Verifies if the allReqs table (requirements data) is empty.
3. **Process Data (if not empty):** If data is available:
	* Removes duplicate entries based on requirement\_guid and vendor\_guid.
	* Extracts the relevant columns (requirement\_guid, vendor\_guid).
	* Adds a column to the table, where each row fetches the detailed requirement data via GetRequirementDetails.
	* Expands the nested details, including name, packages, and multiple nested segments such as questionnaires\_segment, certifications\_segment, audits\_segment, and insurances\_segment.
	* For each segment, further expands and renames columns to extract more detailed information about the artifacts associated with each segment.
4. **Handle Empty Data:** Returns an empty table if no requirements data is available.
5. **Return Result:** Outputs the transformed table containing detailed information about the requirements and their associated segments.

Complexity: Get All Vendors (1 Call) → Get All Requirements (1 Call) → N Requirements → Get Requirements Details →  1 Call per Requirement (N Calls)

### Vendor Lifecyle

**Function:** BitSightSecurityLifecycle.Feed
This function retrieves lifecycle data from a given URL, processes the JSON response, and returns a structured table. It expands certain fields and optionally applies a schema transformation.

|  |
| --- |
| BitSightSecurityLifecycle.Feed = (url as text, endpoint as text, flag as text, optional schema as table) => let source = Web.Contents(url, [ Headers = DefaultVRMRequestHeaders]), json = Json.Document(source), resultsTable = Table.FromList(json, Splitter.SplitByNothing(), null, null, ExtraValues.Error), #"Expanded Column1" = Table.ExpandRecordColumn(resultsTable, "Column1", {"order", "guid", "color\_name", "name", "status", "is\_default"}, {"Column1.order", "Column1.guid", "Column1.color\_name", "Column1.name", "Column1.status", "Column1.is\_default"}), renamed\_cols = Table.RenameColumns(#"Expanded Column1",{{"Column1.guid", "guid"}, {"Column1.name", "name"}}), output = if schema <> null then SchemaTransformTable(renamed\_cols, schema) else renamed\_cols in output;  |

**Steps:**

1. **Fetch Data:** Makes a request to the provided url with the default VRM request headers.
2. **Parse JSON Response:** The response is parsed as a JSON object.
3. **Convert to Table:** The JSON data is converted into a table format using Table.FromList, which transforms the data into rows.
4. **Expand Columns:** Expands the Column1 field to extract multiple nested fields such as:
	* order
	* guid
	* color\_name
	* name
	* status
	* is\_default
5. **Rename Columns:** Renames the expanded columns for better readability:
	* "Column1.guid" → "guid"
	* "Column1.name" → "name"
6. **Apply Schema (Optional):** If a schema is provided, applies the schema transformation using the SchemaTransformTable function. If no schema is provided, the transformed table is returned without schema changes.
7. **Return Output:** Outputs the processed and optionally transformed table with lifecycle data.

Complexity: 1 Call

## Performance Issues

As you can see, as oposed to the CM API implemented in the first version of the connector. Multiple tables are built using endpoints that retrieve data from a single datapoint. For example:

To get All Vendor Details we have to make:

* 1 API Call to get All Vendors List (Returns N Vendors)
* N API Calls to get every detail of every Vendor previously retrieved

**This can lead to long data loading times. It is suggested that users only load the tables that they really need. At the moment, the ‘Questionnaires Answers/Questions’ tables take a considerable amount of time to load due to the need to cross-reference data and the multiple API calls that have to be made.**

# Xpand IT Team

**Data Analytics Engineer**: André Soares

**Delivery Manager**: Luís Velhinho

**Tech Lead:** Carlos Moreira