IBM **Security**
Discover and
Classify

ISDC & TITANIAM SPECTRA INTEGRATION GUIDE

IBM SECURITY DISCOVER AND CLASSIFY

VERSION 3.5.0

TABLE OF CONTENTS

Table of Contents **2**
Overview **3**
Application Flow 3
Prerequisites 4
App Installation, Upgrade & Removal **6**
Installation 6
Upgrade 8
Removal 8
App Usage **9**
Create Usecases using POST Usecases API call 9
Get the List of Existing Usecases Using API Call 10
Get the Stats of Existing Usecases Behavior for Sucessful Encryption Attemps 12
Get the Stats of Existing Usecases Behavior for Failed Encryption Attemps 13

OVERVIEW

Titaniam Server is a Java application that can be configured as one of the following modules:

- Titaniam Encrypted Analytical Vault (further referred as Vault) is a NoSQL datastore. Store your most valuable data and conduct full-featured search analytics while keeping the data encrypted at all times.
- Titaniam Translation Service allows you to encrypt/ decrypt a variety of data (structured, unstructured, binaries). This product can be used to secure sensitive data or to translate data released from other Titaniam products.
- Titaniam Proxy for Object store (referred to as Proxy in this document) allows you to protect an object store.
- Titaniam Elasticsearch Proxy allows you to protect an Elasticsearch node.

The goal of ISDC integration with Titaniam Spectra is to automate encryption of the sensitive files in Amazon S3 buckets.

Table 1: Use cases for ISDC integration with Titaniam Spectra

USE CASE	DESCRIPTION
Automatic encryption of files containing personal data hosted in Amazon S3 bucket	Some S3 buckets may have many files containing plain text sensitive information. After ISDC analyzes the S3 bucket and discovers personal data in files stored there, the integration app will automatically encrypt these files.
File tagging/labeling for encrypted objects in S3 buckets	To provide context for the encrypted files in the S3 bucket, the integration app can assign tags (labels) to the files with useful information. For example, data elements and categories discovered by ISDC in the file.
Reports on workflow behavior, metrics of encrypted files, amount of pass/fail and reasons for failed results	The integration app can keep track on progress of the workflow (how many files Titaniam tried to encrypt and how successful it was), as well as log the affected resources.

APPLICATION FLOW

The integration app is automated. A user manually configures a usecase according to their specific requirements. Then the usecase triggers the following automatic flow:

1. ISDC analyzes the Amazon S3 bucket and discovers personal data in files. These files must be encrypted.

2. Integration app receives the personal data locations (sources) hosted in S3 buckets and processes them according to the usecase scenario.

3. All sources related to the usecase are fetched into the Titaniam tool to manage encryption of the files in S3 bucket.

4. Once the files are encrypted, the integration app receives the status of the encryption request per source.

5. The integration app tags the S3 object: it assigns the data elements and categories found in the object as tags.

Usecase is a rule defined for integration package via API that defines the conditions of the workflow. For example, a user can configure usecases to encrypt only the files with plain text Credit Card Numbers or to encrypt all files with any senstive information in a specific bucket. These usecases can be dynamically enabled and disabled.

A usecase has a set of mandatory and optional configurations that define the workflow:

- Data Elements are a list of ISDC data element names that trigger the usecase execution. The data element names are dynamically updated according to ISDC settings. The integration app supports the out-of-box and custom data elements.
- Source URL is a string that filters the discovery results for presence of specific bucket to work with.

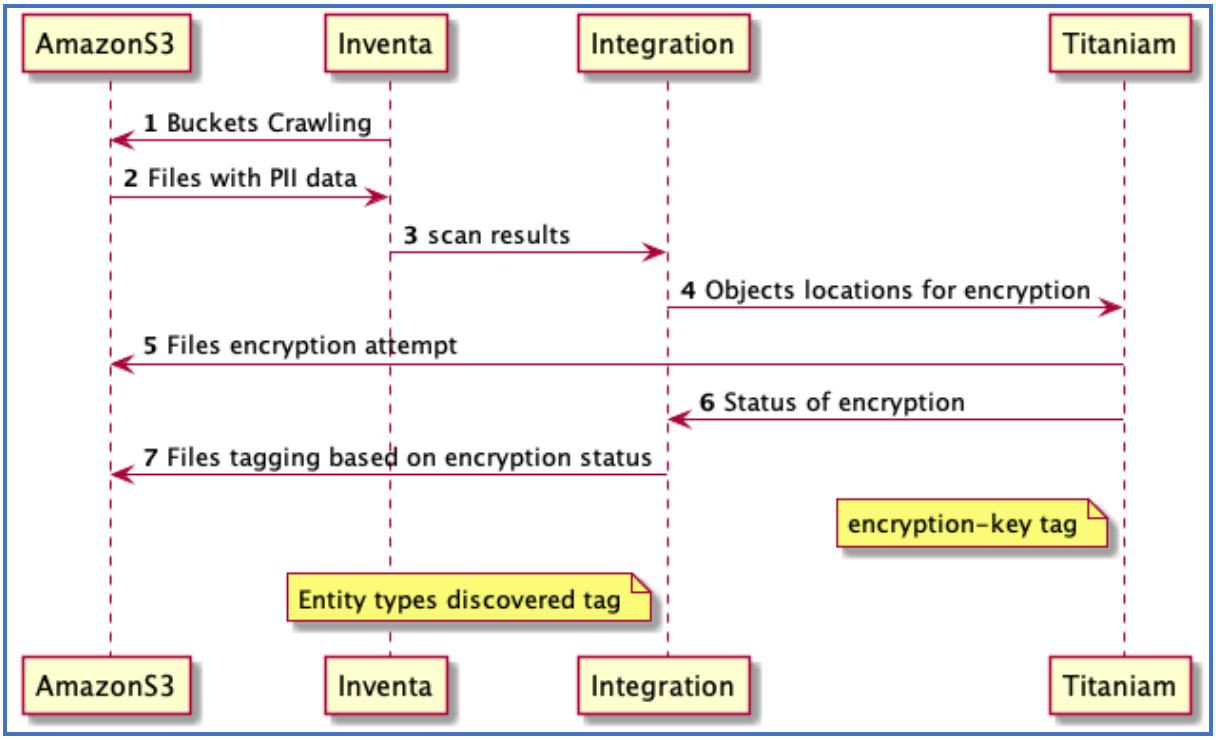

Figure 2: Flow of ISDC integration with Titaniam Spectra

PREREQUISITES

1. Titaniam Spectra installed:

- For Docker deployment, follow the Production Installation Using Docker on EC2 Instance.
- For K8s deployment, follow the Production Installation Using Kubernetes.

2. ISDC v3.4+ installed and having access to Titaniam Spectra instance:

```
telnet <titaniam_host> <titaniam_port>
```

3. AWS access key ID and secrets generated with the following permissions for the targetted S3 buckets (or all of them depending on usecase):

- s3:ListBucket
- s3:*Object
- s3:GetBucketVersioning

- s3:GetObjectTagging
- s3:PutObjectTagging

4. Integration application installed. Follow the installation steps.

APP INSTALLATION, UPGRADE & REMOVAL

INSTALLATION

Integration package is provided on demand. It can be installed on your VM, local machine or cluster node. To deploy the project, perform the following steps:

1. Extract the package directory using one of commands below depending on the archive extension (zip, tar, tar.gz):

```
unzip titaniam-integration.zip
```

or

```
tar -xvf titaniam-integration.tar
```

or

```
tar -xvf titaniam-integration.tar.gz
```

2. Change directory to extracted folder:

```
cd titaniam-integration-master
```

3. Copy the ./env.compose.example file, rename it to env.compose:

```
cp .env.compose.example .env.compose
```

4. Fill the file with data. See the sample configuration below:

```
Sample .env.compose file:

APP_NAME=titaniam-integration-1
APP_HOST=10.192.192.117
JWK_JWT=secret

KAFKA_HOST=10.192.192.131
KAFKA_PORT=30828
KAFKA_USERNAME=kadmin
KAFKA_PASSWORD=kafkapassword
SECURITY_PROTOCOL=SASL_SSL


TITANIUM_HOST=18.206.252.64
TITANIUM_PORT=8080
```

```
TITANIUM_SCHEME=http

AWS_ACCESS_KEY_ID=AKIARMCLMCOXUUXAT7ED
AWS_SECRET_ACCESS_KEY-
Y=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

POSTGRES_HOST=integration_postgres_splunk
POSTGRES_PORT=5432
POSTGRES_DB=integration_db
POSTGRES_USER=postgres
POSTGRES_PASSWORD=postgres
POSTGRES_POOL_MAX_CONN=3
```

5. You need to change APP_HOST, KAFKA_HOST, KAFKA_PORT and titaniam details (host, port scheme and AWS access key).

5.1. To learn the host and port of Kafka, run this command:

```
kubectl get svc -n {YOUR_NAMESPACE} kafka-cm-tls-0-external
```

Value under EXTERNAL IP is the KAFKA_HOST. The KAFKA_PORT is the value next to port 9094: 9094:<value>.

5.2. KAFKA_HOST is the IP of appliance, which you can find as follows:

```
kubectl get nodes -n {YOUR_NAMESPACE}
```

In the output, use the node IP address.

5.3. App_host is hostname or IP address of ISDC you wich to import data from.

6. Start the stack:

```
docker-compose up -d --build
```

7. After the stack is started you should be able to reach the API swagger of the integration on https://<APP_HOST>:5005/docs.

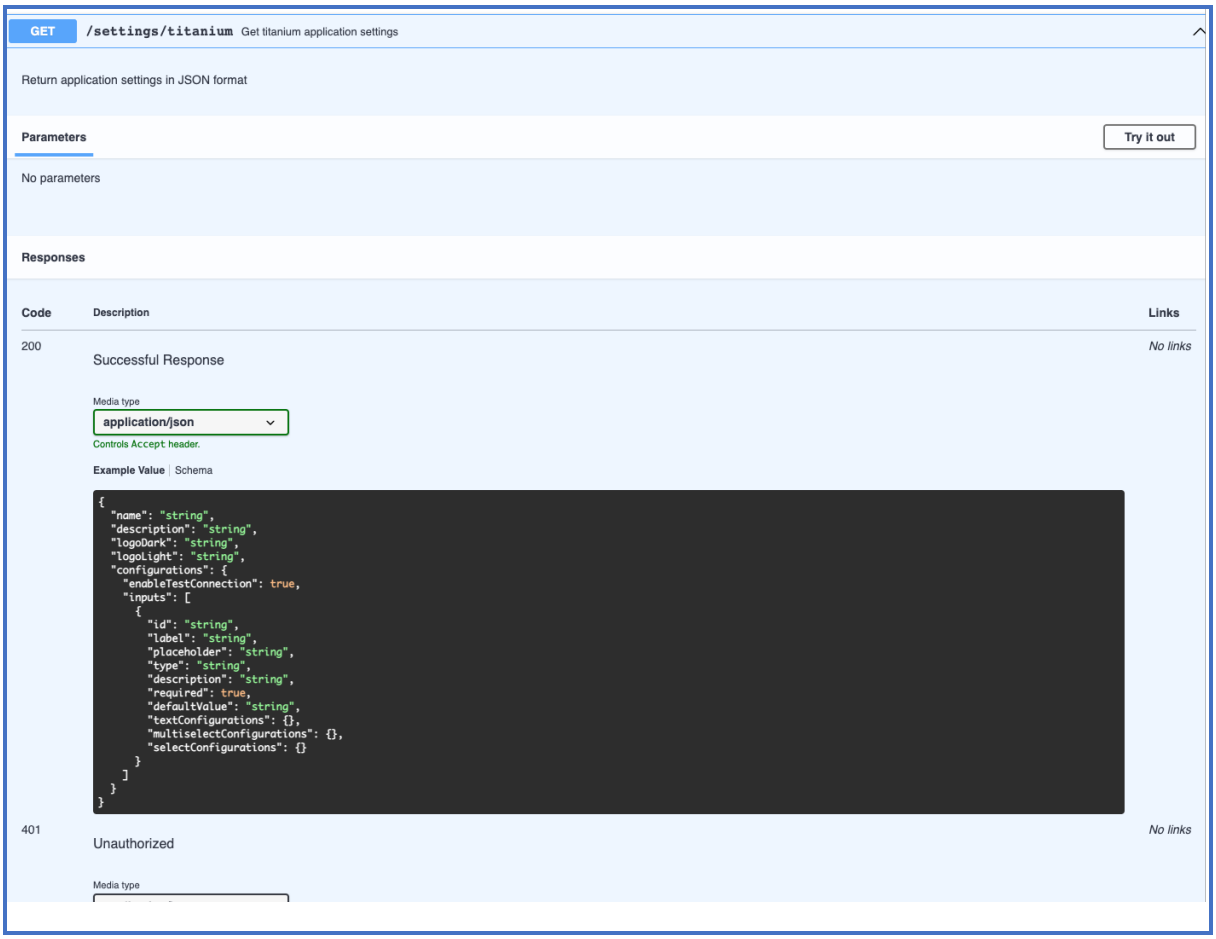

Figure 1: API swagger of the integration

UPGRADE

To install newer version of application, the process is identical to installation. The docker volumes removal may be required, if explicitly specified.

REMOVAL

1. To delete the application, stop the containers from the directory with deployment files:

```
cd titaniam-integration-master
docker-compose down
```

2. Remove the docker volumes created for the project. Volumes can be listed using command:

```
docker volume rm ${docker volume ls -q |grep titaniam}
```

APP USAGE

CREATE USECASES USING POST USECASES API CALL

Sample request

```
curl -X 'POST' \
  'https://<APP_HOST>:5005/usecases' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
  "name": "Test Usecase 1",
  "description": "Describe what this use case should cover",
  "data_elements": [
    'CC_NUMBER',
    'ACCOUNT_ID',
    'PHONE_NUMBER'
  ],
  "source_url": "s3://",
  "action": "encrypt",
  "active": true
}'
```

Sample response

Success

```
{
  "success": true
}
```

Fail (for example, incorrect data elements input syntax)

```
{
  "detail": [
    {
      "loc": [
        "body",
        116
      ],
      "msg": "Expecting value: line 5 column 5 (char 116)",
      "type": "value_error.jsondecode",
      "ctx": {
        "msg": "Expecting value",
        "doc": "{\n  \"name\": \"Test Usecase 1\",\n  \"description\": \"Describe what this use
case should cover\",\n  \"data_elements\": [\n    'CC_NUMBER',\n    'ACCOUNT_
ID',\n    'PHONE_NUMBER'\n  ],\n  \"source_url\": \"s3://\",\n  \"action\":
\"encrypt\",\n  \"active\": true\n}",
```

```
      "pos": 116,
      "lineno": 5,
      "colno": 5
    }
   }
  ]
}
```

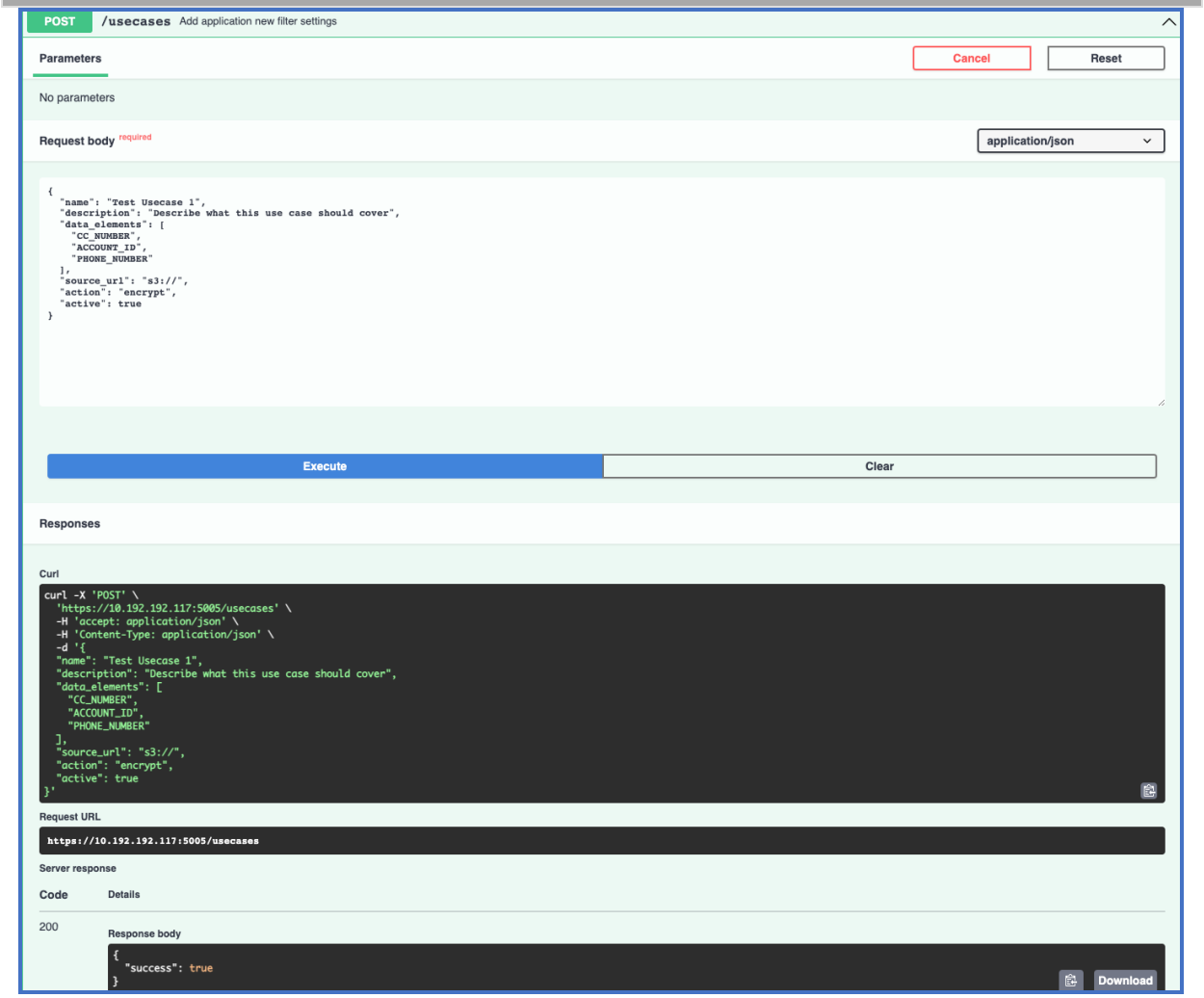

Figure 1: Example of usecase creation

GET THE LIST OF EXISTING USECASES USING API CALL

Sample request

```
curl -X 'GET' \
 'https://10.192.192.117:5005/usecases/titanium' \
 -H 'accept: application/json'
```

Sample Response


```
[
  {
    "id": 67,
    "name": "Titaniam Use case",
    "description": "string",
    "action": "encrypt",
    "active": true,
    "counter": 24307,
    "success_counter": 6,
    "failed_counter": 24297
  }
]
```


Table 2: Response parameters for existing usecases request

PARAMETER	DESCRIPTION
id	Unique identifier of usecase entity.
name	Usecase alias.
description	Usecase description.
action	Type of action on source location. For Titaniam, only *encrypt* is available.
active	Indicates whether the usecase is processing data. Options: true; false.
counter	Metric for the number of source records processed in total. Numeric.
success_counter	Number of files that were successfully encrypted.
failed_counter	Number of files that the system failed to encrypt due to various reasons.

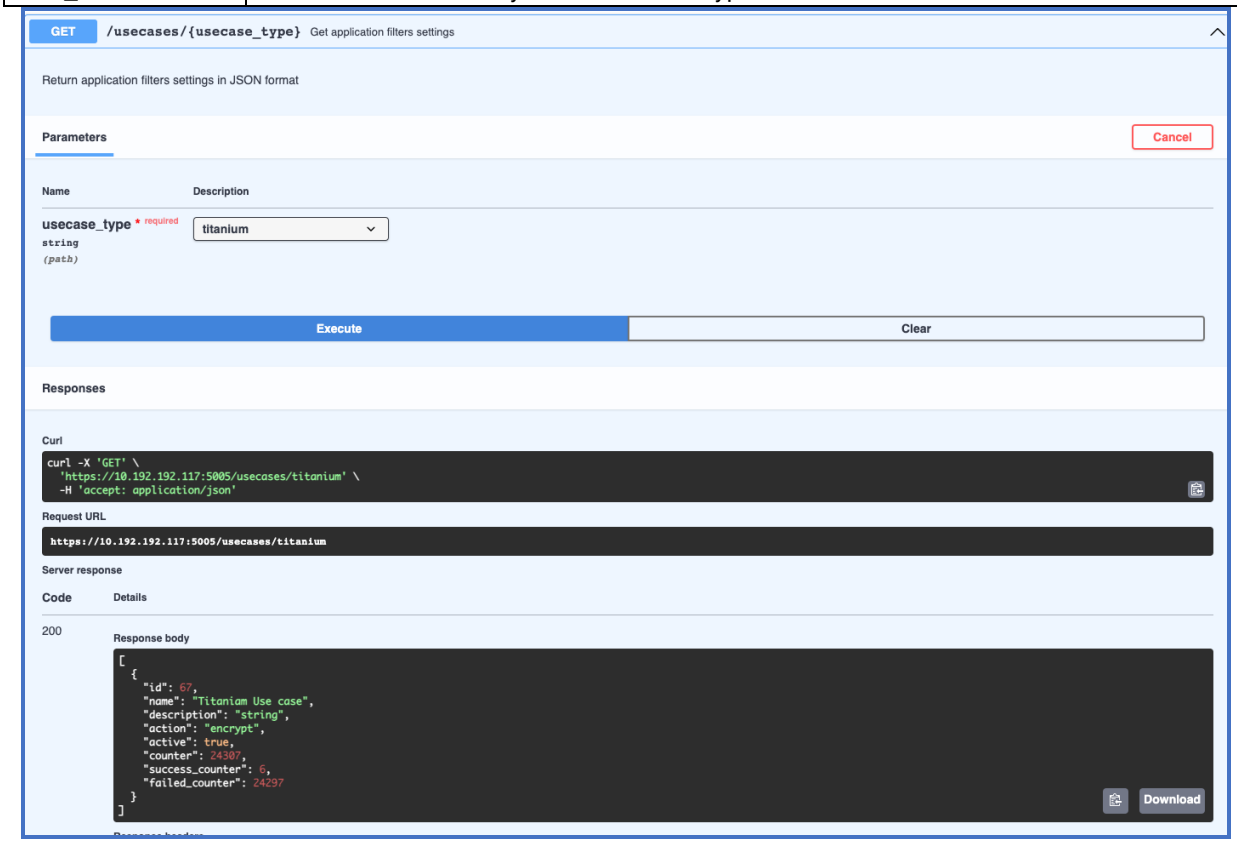

Figure 3: Example of usecase list request

GET THE STATS OF EXISTING USECASES BEHAVIOR FOR SUCESSFUL ENCRYPTION ATTEMPS

Sample Request

```
curl -X 'GET' \
  'https://10.192.192.117:5005/get_encrypted_files?lim-
it=1000&offset=0&encrypted=true' \
  -H 'accept: application/json'
```

Sample Response

```
[
  {
    "encrypted": true,
    "error_description": null,
    "id": 379,
    "path": "dev-titaniam-integration/Brazil250pavlo13.csv"
  },
  {
    "encrypted": true,
    "error_description": null,
    "id": 395,
    "path": "dev-titaniam-integration/Brazil250pavlo27.csv"
  },
  ...
  ...
  ...
]
```

Table 4: Response parameters for sucessful encryption attemps

PARAMETER	DESCRIPTION
encrypted	Current state of file encryption request. Options: true; false.
error_description	Applicable if the encryption attempt failed.
id	Unique identifier of the resource to encrypt.
path	Full path to resource.

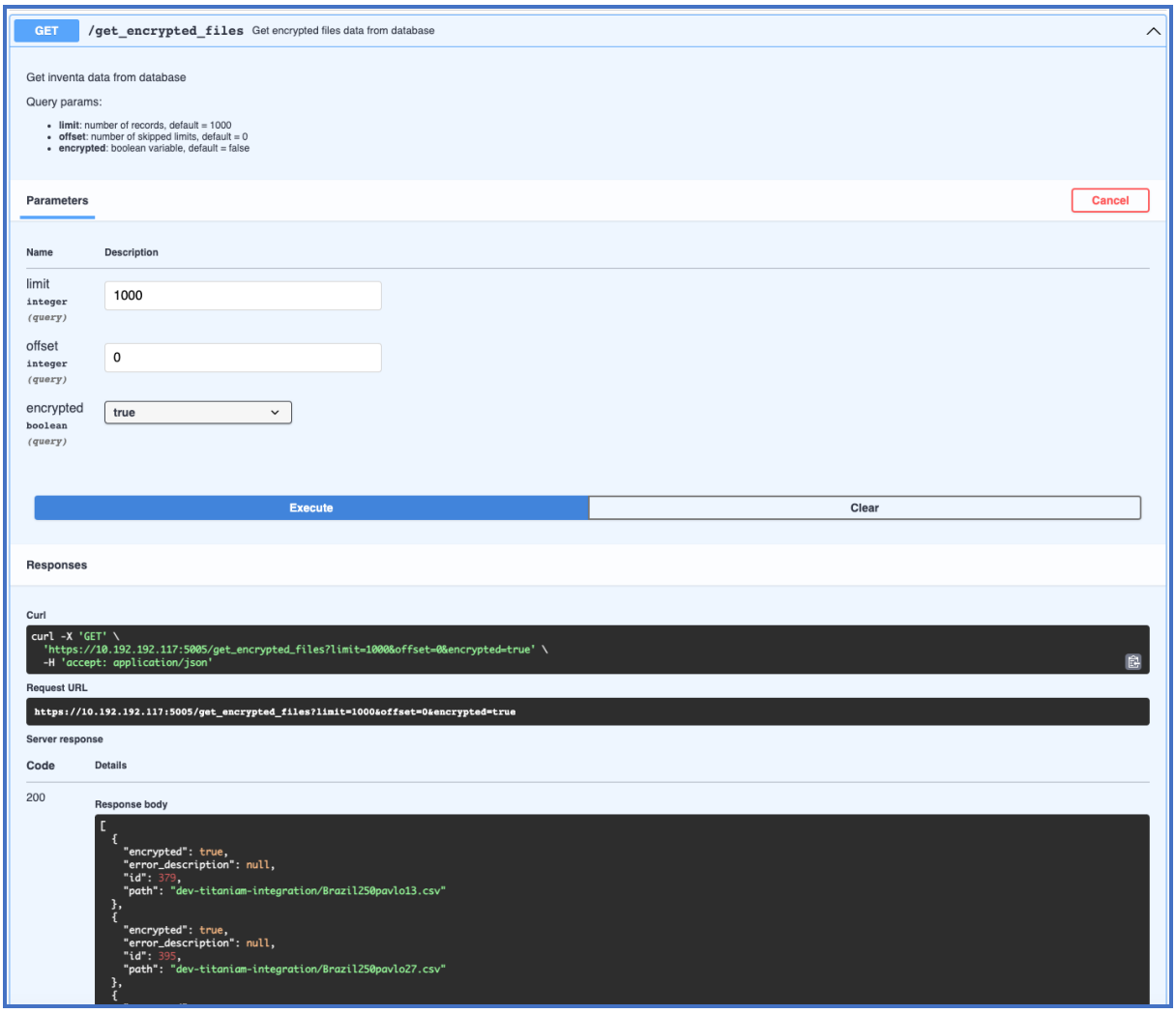

Figure 5: Example of stats for successful encryption attemps

GET THE STATS OF EXISTING USECASES BEHAVIOR FOR FAILED ENCRYPTION ATTEMPS

Sample Request

```
curl -X 'GET' \
  'https://10.192.192.117:5005/get_encrypted_files?lim-
it=1000&offset=0&encrypted=false' \
  -H 'accept: application/json'
```

Sample Response

```
[
  {
```

```
    "encrypted": false,
    "error_description": "The bucket you are attempting to access must be addressed
using the specified endpoint. Please send all future requests to this endpoint. (Service:
S3, Status Code: 301, Request ID: WG4XXJ2GYVV6Q6Y3, Extended Request ID:
oQ5bVaONJ6S+FvAOaNtSVE/KMt3ip4cMEHUig17iiqEEh0SqJQ76/s0HACxqkOVUBQ-
hwhhmYU28=)",
    "id": 1,
    "path": "1touch-athena-results/0e52ae62-f9ca-4564-9c69-aace234dc81d.csv"
  },
  {
    "encrypted": false,
    "error_description": "The bucket you are attempting to access must be addressed
using the specified endpoint. Please send all future requests to this endpoint. (Service:
S3, Status Code: 301, Request ID: T6CXKJN2XY33196R, Extended Request ID:
lCeOJSmuADe76Gmg-
gtxDdZV73K6L36HEOkooS3OpslOYqeHLqyVBf93/mONuWrp+GJ16pbA2r0Q=)",
    "id": 2,
    "path": "1touch-athena-results/0f645974-d011-41b2-8f0c-edf947b8e57a.csv"
  },
  {
    "encrypted": false,
    "error_description": "The bucket you are attempting to access must be addressed
using the specified endpoint. Please send all future requests to this endpoint. (Service:
S3, Status Code: 301, Request ID: 3CRC9BWH73GK4V67, Extended Request ID:
oU6ed8xUtGwNDqM2Yx2uCAY3Z4L4BtTVb2CdyUo1xK468d3JLjMTEk74NU4iEQv4X-
Dcy3vGdlc0=)",
    "id": 3,
    "path": "1touch-athena-results/09d1a304-8650-494f-9850-2956b2d410d5.csv"
  },
  ...
  ...
  ...
]
```


Table 6: Response parameters for failed encryption attemps

PARAMETER	DESCRIPTION
encrypted	Current state of file encryption request. Options: true; false.
error_description	Reason for failed encryption attempt.
id	Unique identifier of the resource to encrypt.
path	Full path to resource.

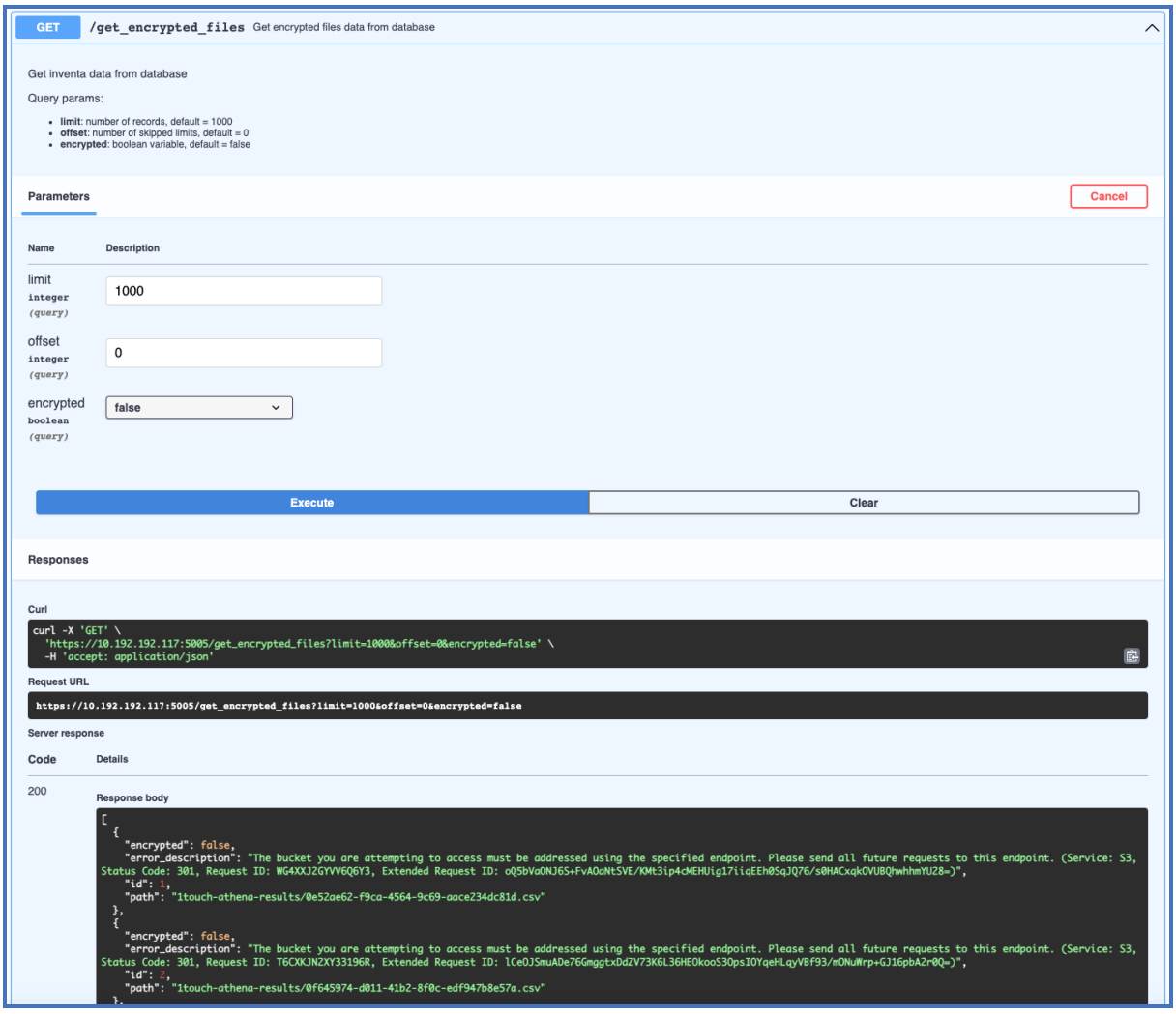

Figure 7: Example of stats for failed encryption attemps

Result of the operations above is an encrypted file on S3 bucket and assigned tags specifiying the context around encryption:

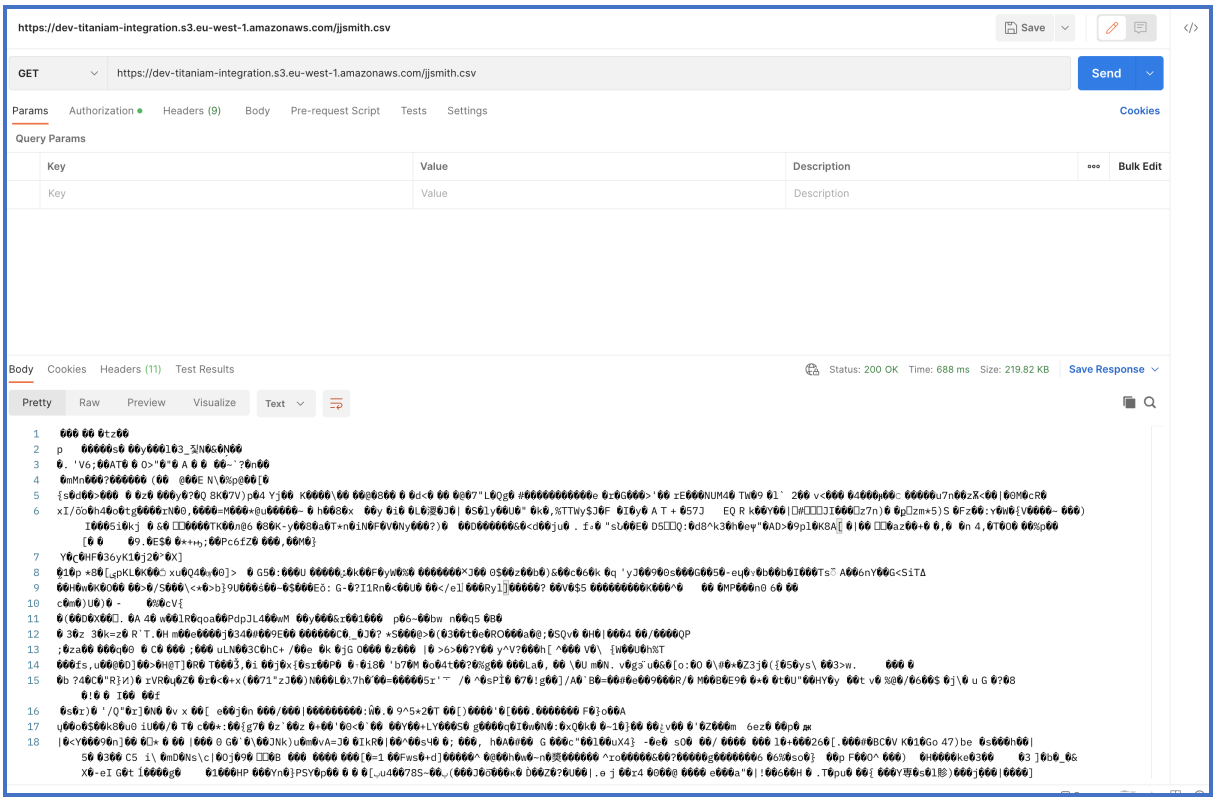

Figure 8: Encrypted file output

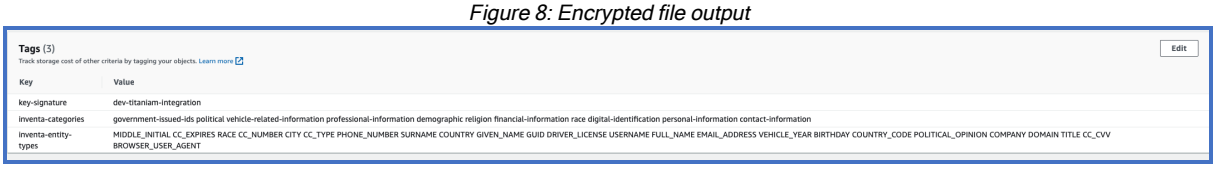

Figure 9: S3 resource tagging