VMRAY

VMRAY PLATFORM

API Programmer Guide

Version 2026.2.1

Copyright

Disclaimer

The contents of this documentation depict a guideline for using and operating VMRay software. It is possible that this documentation might contain inaccuracies or errors. Although VMRay attempts to ensure the completeness and accuracy of this documentation, there is no guarantee as to the correctness or accuracy of the documentation. In the event of inaccuracies, please inform VMRay.

The information contained herein may be changed without prior notice.

VMRay assumes no responsibility for errors or omissions in this document. VMRay does not warrant the accuracy or completeness of the information, text, graphics, links, or other items contained within this material. This document is provided without a warranty of any kind, either expressed or implied, including but not limited to the implied warranties of merchantability, fitness for a particular purpose or non-infringement.

VMRay shall not be liable for damages of any kind including, and without limitation, direct, special, indirect or consequential damages that may result from the use of these materials including example code. This limitation shall not apply in cases of intent or gross negligence. The statutory liability for personal injury and defective products is not affected. VMRay has no control over the information that may be accessed through the use of hot links contained in these materials and does not endorse the use of third-party web pages, nor provide any warranty whatsoever relating to third-party web pages.

Several software products marketed by VMRay and its distributors contain proprietary software components of other software vendors.

Apple® and Mac® are registered trademarks Apple Inc., registered in the U.S. and other countries.

Microsoft®, Microsoft Access®, Microsoft Excel®, Microsoft Office®, Microsoft Powerpoint®, Microsoft Visio®, Microsoft Windows®, Microsoft Word®, and the respective logos are registered trademarks of Microsoft Corporation, Redmond, WA.

Hangul Office, Hanword, Hancell and Hanshow are trademarks of Hancom Inc., Seongnam, South Korea.

Linux® is the registered trademark of Linus Torvalds in the U.S. and other countries.

Adobe®, Adobe Acrobat®, Adobe Acrobat Reader®, Adobe PostScript, and the Adobe logo are either trademarks or registered trademarks of Adobe Inc. in the United States and/or other countries.

HTML, W3C®, XHTML, and XML are trademarks or registered trademarks of W3C, World Wide Web Consortium, Massachusetts Institute of Technology (MIT).

VirusTotal® is the registered trademark of Chronicles Security Ireland Limited in Dublin, Ireland. All other product and service names mentioned are the trademarks of their respective companies. Data contained in this document serve as informational purposes only. National product specifications may vary.

Table of Contents

1 About this Programmer Guide 3
1.1 Scope 3
1.2 Audience 3
1.3 Purpose 3
1.4 Related Documentation 3

2 Overview 4
2.1 API Functionalities per Product 4
2.2 Understanding the Interfaces to the Platform 4
2.3 Viewing the Interfaces on the Submissions Page 7

3 Creating API Calls 9
3.1 Understanding the Structure of API Calls 9
3.1.1 HTTP Methods 9
3.1.2 The Host Endpoint 9
3.1.3 Function Calls 10
3.2 Authenticating with an API Key 10
3.3 Creating an API Key 11
3.4 Executing Your First Call 14

4 Using Functions 16
4.1 Understanding Function Categories 16
4.2 Understanding Function Classes 16
4.3 Get APIs 16
4.3.1 Using Filters 17
4.3.2 Understanding the Common Query Parameters 17
4.3.3 Circumventing Result Truncation 18
4.3.4 Interval Searching 18
4.4 Creation APIs 19
4.5 Update APIs 19
4.6 Deletion APIs 20

5 Understanding Parameters 21
5.1 File URL Support (On Premises only) 22
5.1.1 Referencing a Previously Submitted Sample 22
5.1.2 Using a Local Sample Upload Directory 22

6 Understanding the Result Format 25
6.1 Understanding Result Caching 25
6.2 Understanding Result Truncation 26

7 Looking at a Simple Example 27

8 Tutorial: Submitting Sample Files and URLs 29
8.1 Getting System Information with GET /info 29
8.2 Submitting a Sample with POST /sample/submit 30
8.2.1 Using the Console to Submit a File 30
8.2.2 Using an API Call to Submit a File 35
8.2.3 Looking at the API Submission in the Console 36

8.3 Submitting a URL with POST /sample/submit 39
9 Tutorial: Getting Submission and Sample Data 41
9.1 Getting Submission Data with GET /submission 41
9.2 Getting Sample Information with GET /sample 44
10 Tutorial: Submitting Archives (Recursive Analysis) 46
10.1 Understanding Recursive Analysis 46
10.2 Looking at a ZIP File 47
10.3 Submitting a ZIP Archive File using the Console 47
10.4 Understanding the Three Archive Actions 53
10.5 Submitting an Archive with POST /sample/submit 54
11 Tutorial: Understanding Verdict and Report Quotas 57
11.1 Understanding the Pricing Plans 57
11.2 Viewing your Quota in the Console 57
11.2.1 Cloud 58
11.2.1.1 Account Managers 58
11.2.1.2 Administrators 58
11.2.1.3 Standard Users 59
11.3 Checking Quota Usage per Submission 60
11.4 Defining Quota for API Keys 60
12 Tutorial: Understanding Analysis Caching 61
12.1 Submitting with Analysis Caching Disabled 61
12.2 Submitting with Smart Caching Enabled 64
13 Tutorial: Understanding Webhooks 67
13.1 Understanding the Webhook Parameters 67
13.2 Submitting a Sample with Activated Webhook Notification 68
13.3 Webhook Specifics on Cloud 68
13.4 Understanding the Webhook Notification Format 68
14 Installing and Using the Integration Kit 70
14.1 An Overview of the Integration Kit 70
14.2 Installing the Integration Kit 70
14.3 Reviewing the Python Example Code 72
14.4 Understanding the Two Required Calling Parameters 73
15 Using the API Client Examples 74
15.1 Submission Programs 74
15.2 Download Programs 74
15.3 Job Programs 74
15.4 Maintenance Programs 75
16 Using the Integration Kit Examples 76
16.1 Understanding the Five Example Code Files 76
16.2 Looking at an Example Call 76
16.3 Getting Help 77

1 About this Programmer Guide

Welcome to the *API Programmer Guide.*

1.1 Scope

This *Guide* provides basic information about using the VMRay Platform REST API.

This *Guide* is for both Cloud and On Premises customers.

1.2 Audience

This *Guide* is for programmers who are comfortable writing API calls.

1.3 Purpose

The VMRay Platform products - DeepResponse, FinalVerdict* and TotalInsight - provide a world-class set of tools for malware detection and analysis. The purpose of this Guide is to enable you to interact with the VMRay products using our REST API.

*Note that the FinalVerdict product is available for Cloud accounts only.

1.4 Related Documentation

More detailed information about all function calls available through the REST API is available in:

- *Cloud API Reference* (for Cloud customers only)
- *On Premises API Reference* (for On Premises customers only)

Information about the summary.json v2 and other reports and files generated during analysis is provided in the:

- *Analysis Results Reference v2*

Information about the legacy summary.json v1 and other reports and files generated during analysis is provided in the:

- *Analysis Results Reference v1* (Legacy)

2 Overview

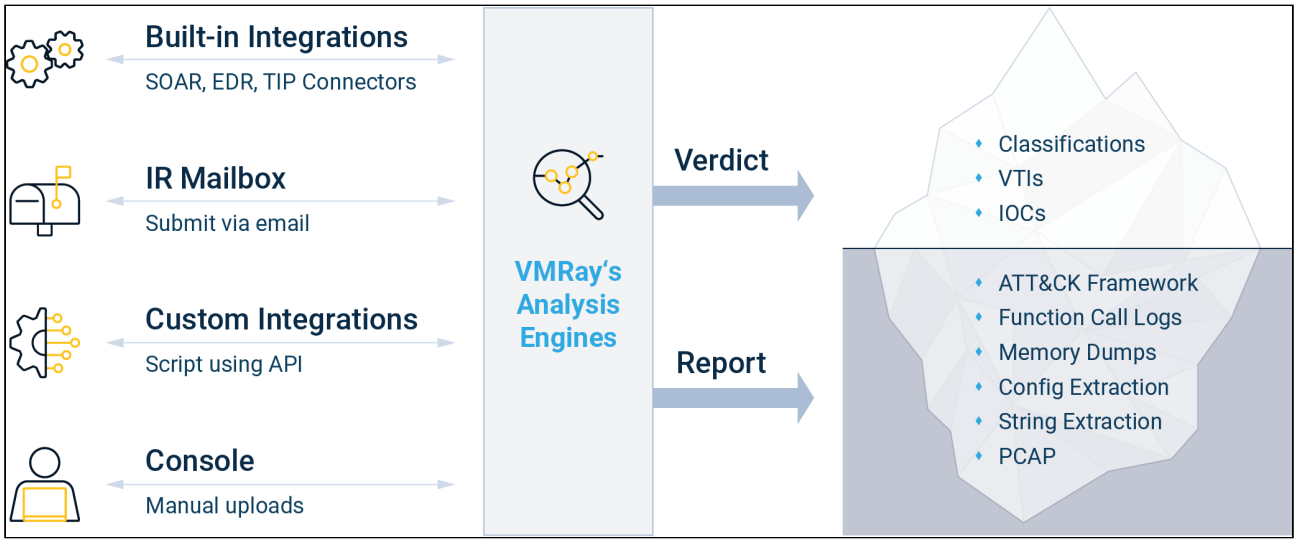
The REST API is one of four interfaces available for interacting with the VMRay Platform. This chapter describes the VMRay Platform architecture in order to make it clear how the REST API fits into the overall picture, starting with the four interfaces to the Platform. After that, this chapter shows you how to use the Console with the REST API, for example, to view samples that you have submitted through the API.

2.1 API Functionalities per Product

VMRay products - DeepResponse, FinalVerdict and TotalInsight have different capabilities. Consult the Appendix - Supported Product Features in the *On Premises* or *Cloud Onboarding Guide* to have a complete view of features available in each product. Concerning the APIs, we support inbound and outbound APIs. Note that the only inbound API is the sample submission endpoint, which will be indicated further in this document. The remaining APIs are outbound. Since each product's capabilities vary, DeepResponse does not support inbound API functionality, so the DeepResponse user will not be able to submit the samples via API.

2.2 Understanding the Interfaces to the Platform

The diagram below illustrates the basic architecture of the VMRay Platform products and the four interfaces that are supported:



The interfaces depicted above are described below:

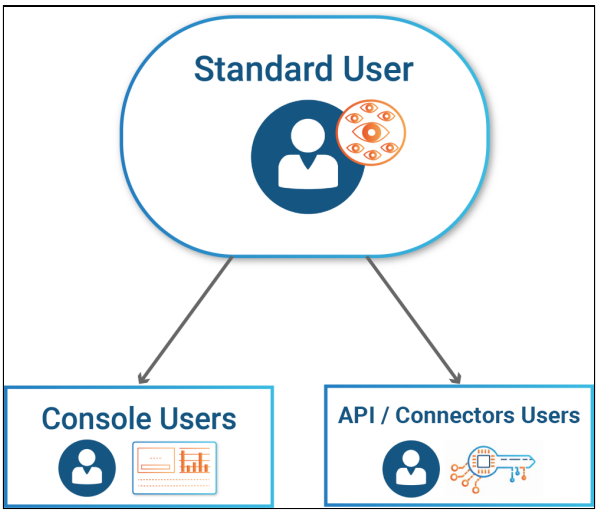
1. **Built-in Integrations**, which are software solutions built into VMRay Platform that supplement our product to deepen the insight into the malware. Our Cloud customers can integrate with Virustotal; On Premises customers can additionally create Custom Reputation Plugin and Custom Built-in AV Plugin.

2. The **Console** which is the standard GUI to interact with.

3. The **IR Mailbox** enables anyone in your company to send suspicious emails to a central email address set up by you, the Account Manager. These emails are then automatically examined for malware. The IR Mailbox helps you build a comprehensive Incident Response (IR) strategy around our product. The *Cloud Account Manager Guide* describes how to set up an IR Mailbox, and the *Console User Reference* describes how to use the IR Mailbox. The IR Mailbox option is not supported in DeepResponse product.

4. **Custom Integrations** allow you to access the functionality of our products programmatically. You can do it by using the REST API. The API Programmer Guide and the Cloud API Reference have complete information about using the REST API. Note that Custom Integrations via API are available for FinalVerdict and TotalInsight.

It is important to note that users who want to use the REST API are assigned the **Standard User** role (by the Administrator or Account Manager of your VMRay Platform), which enables you to log in and use the Console, and most importantly, gives you the ability to set up your own API keys in the Console. **Console Users**, also have the **Standard User** role assigned to them, but they do not necessarily use the REST API. The two user types and one role are depicted below:

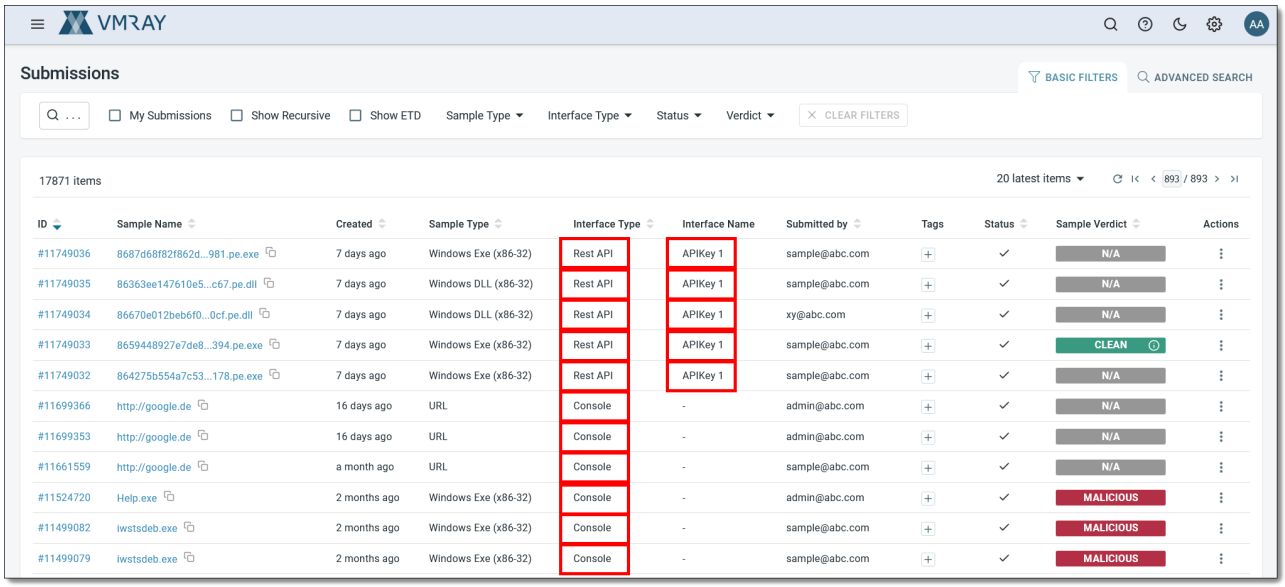


If you have not yet received credentials to login to the platform, talk to your Account Manager (cloud customers) of your Administrator (On Premises customers).

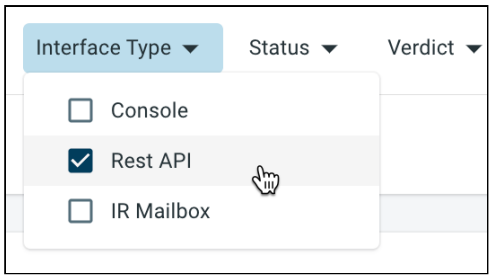
One of the first things you should do after logging in to the Platform is take a look at the Submissions page, which is described in the next section.

2.3 Viewing the Interfaces on the Submissions Page

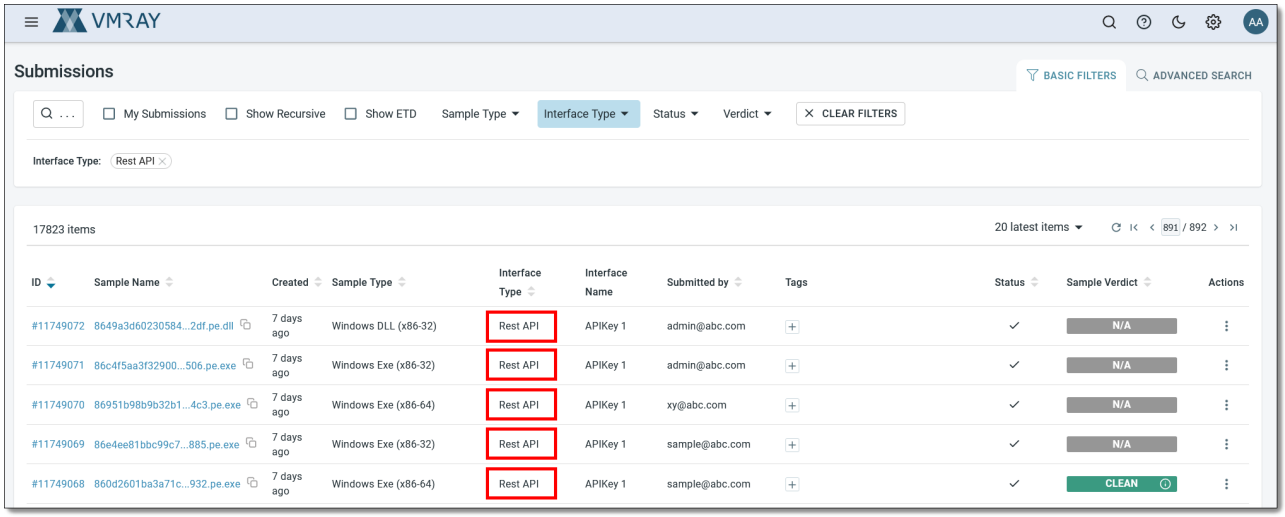
For all submissions to VMRay products, you can see the interface used in the **Interface** column of the **Submissions** page, as shown below, where the first two submissions are from the **Console**, while the next four are from the **REST API**:



When the Interface is **REST API**, the actual API key name displays in the next column, **APIKey 1** in the example above. To see only submissions from the **REST API**, select it from the **Interface Type** drop-down near the top of the page:



After selecting the **Interface Type**, you will see, in this case, only submissions from the REST API:



As you start to submit samples using the REST API, you can review them on the Submissions page.

3 Creating API Calls

This chapter describes the basic format of your API calls and how to authenticate them. This chapter also has information about permissions and user impersonation.

Accessing REST API with DeepResponse Essential and Professional is unavailable.

3.1 Understanding the Structure of API Calls

Here is an example of a typical REST API call:

```
GET "https://eu.cloud.vmray.com/rest/system_info"
```

This call retrieves basic system information using the **system_info** endpoint. Let's look in more detail at its three elements.

3.1.1 HTTP Methods

As shown above, your calls should always start with an HTTP method. The API supports three HTTP methods:

- **GET** to retrieve data
- **POST** to store data
- **DELETE** to delete data

3.1.2 The Host Endpoint

Next, your call should use the appropriate HTTPS endpoint of the host that you are using. If you are using the Cloud version of the VMRay Platform, you should use:

- https://eu.cloud.vmray.com - if you are based in Europe
- https://us.cloud.vmray.com - if you are based in the United States

If you are using the On Premises version of the VMRay Platform, then you must use the URL that has been set up by your Administrator.

In the code examples provided in the available *API References*, the host endpoint is always shown as: **<YOUR_HOSTNAME_HERE>**.

3.1.3 Function Calls

Next, all REST API calls must start with the **/rest/** prefix. After which, you specify the actual function, such as **system_info,** so that a complete call looks like this:

```
GET "https://eu.cloud.vmray.com/rest/system_info"
```

Leaving some trailing slashes in your URLs might result in an error.

All of the function calls that are available are described in the *API References*. The *Cloud API Reference* is for Cloud customers. The *On Premises API Reference* is for On Premises customers.

3.2 Authenticating with an API Key

The VMRay Platform uses API keys to provide authentication for calls to the REST API. It is not possible to call an API function without providing a valid API key.

API keys must be provided in the **HTTP Authorization** header using the **api_key** parameter, followed by the plain text API key. Here is a typical call that uses **curl**:

```
curl -H "Authorization: api_key ZCJxiTKEP3F5tDSGttPQ2FrBYQ67pJUe" -X GET "https://
eu.cloud.vmray.com/rest/system_info"
```

The REST API is currently only accessible using encrypted HTTPS connections. This prevents leakage of the API key, which must be provided in plain text.

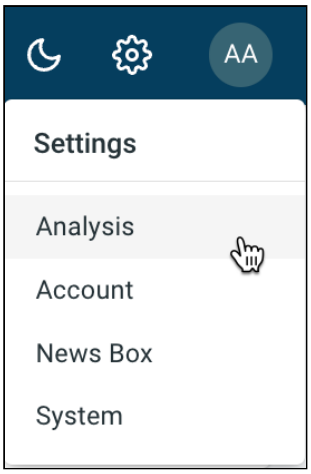
If you plan to use the API with HTTP by modifying the **nginx** configuration, then you must be aware of this shortcoming.

3.3 Creating an API Key

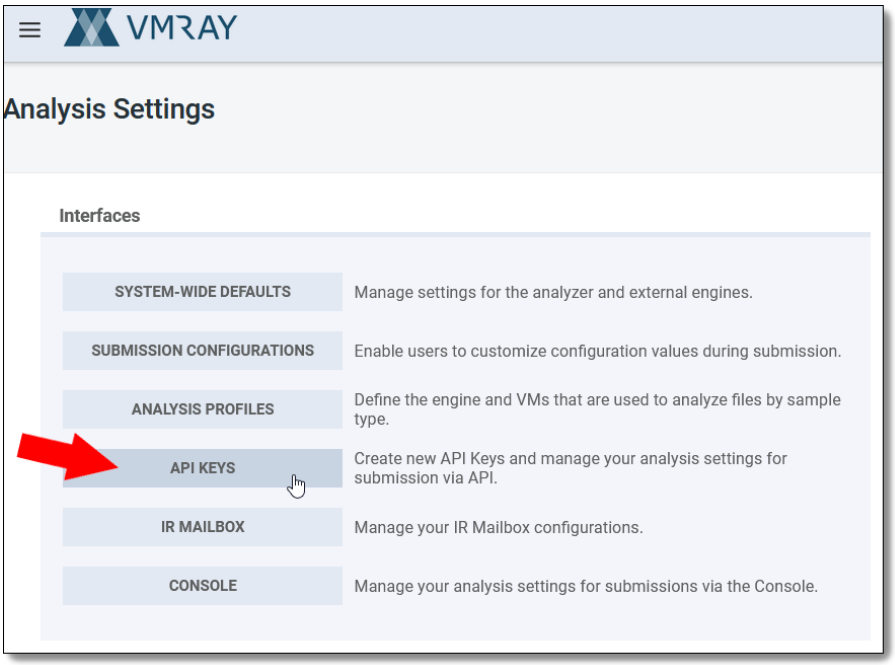
Creating API keys in the Console with DeepResponse Essential and Professional is unavailable.

To create your own API key:

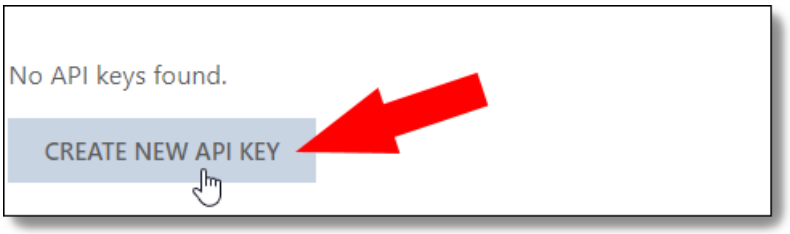
1. Log in to the Console of the VMRay Platform. Consult your Account Manager (on Cloud) or Administrator (On Premises) if you do not yet have credentials.
2. Select **Analysis** from the Settings menu:



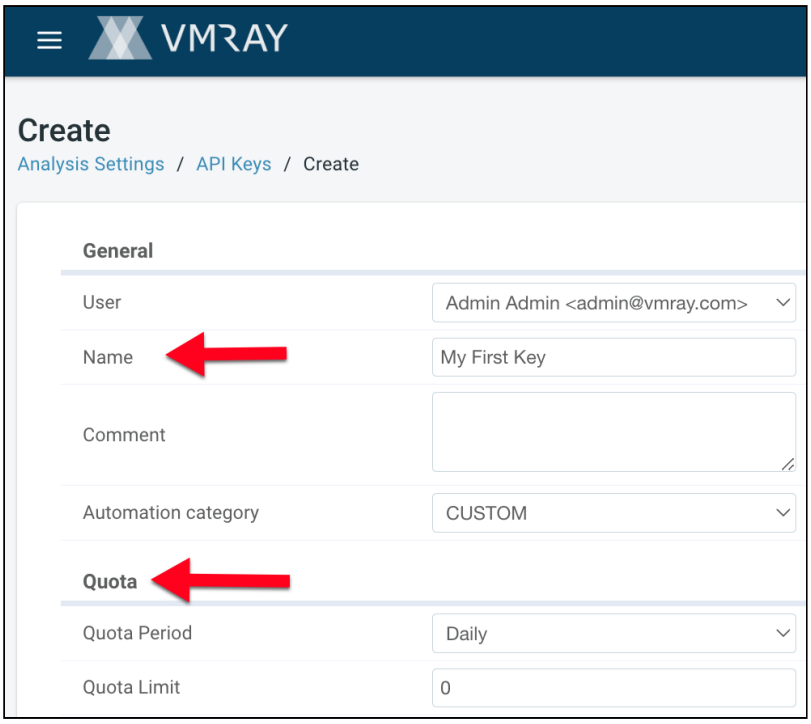
3. Select the **API KEYS** button which is the first one that displays in the **Interfaces** area:



4. Click the **CREATE NEW API KEY** button that displays:

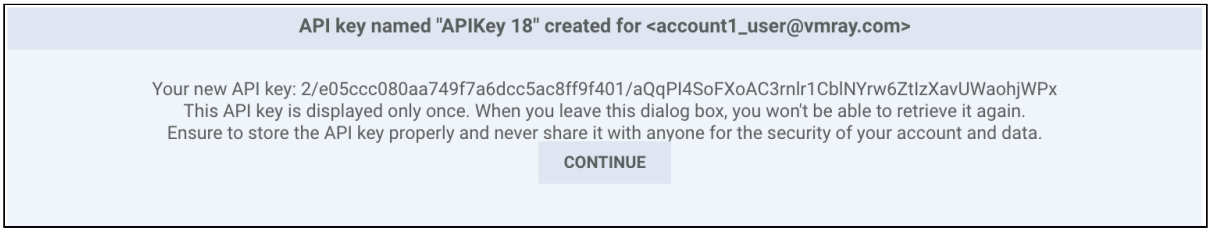


5. Fill in the **Name** entry field, as in the example below, where **My First Key** is used. Specify the **Quota** settings such as **Quota Period** and **Quota Limit**:



6. Further down this page, there are a variety of Analysis configuration options, but leave the defaults for now and click the **SAVE** button at the bottom of the page and your API Key is generated. Note that the new API Key you just created is displayed **only once** and once you press the **CONTINUE** button, you will not be able to retrieve it again.

Make sure to copy this key and store it securely, so you can use it in your API calls.



All of the Analysis configuration options are described in the *Console User Reference*.

7. You're back on the API Keys page that displays all of your API keys. You'll notice the Secret key-ID is displayed instead of the API secret key in plain text . You can **Edit** or **Delete** the API Keys:

The "**-**" below Secret ID is displayed for the legacy API Key format that we used before the 2023.4.0 release.

8. Choose **Edit** if you want to modify the Analysis configuration options, but before you do so, see the *Console User Reference* because it describes each setting in detail. The defaults work fine so you can leave them as is for now.

In the code examples provided in the available *API Reference* documents, the API key is always shown as: **<YOUR_APIKEY_HERE>**.

For On Premises customers, it is possible to programmatically generate API keys. Consult the *On Premises API Reference* for more information.

3.4 Executing Your First Call

Now that you have your API key, you can run your first API call, as in this example below, which uses the API key generated above, and the European Cloud HTTP endpoint:

```
curl -H "Authorization: api_key ZCJxiTKEP3F5tDSGttPQ2FrBYQ67pJUe" -X GET "https://
eu.cloud.vmray.com/rest/system_info"
```

The **system_info** function call returns basic system information in JSON format, as in this example:

```
{
 “data”: {
  “api_items_per_request”: 100,
  “file_param_http_scheme_enabled”: true,
  “max_api_items”: 1000,
  “version”: “4.3.0",
  “version_major”: 4,
  “version_minor”: 3,
  “version_revision”: 0,
  “webif_alias”: “EU Cloud”,
  “webif_base_url”: “https://eu.cloud.vmray.com”,
  “webif_max_sample_size”: 209715200,
  “webif_max_upload_size”: 209715200
 },
 “result”: “ok”
```

The **data** section displays various system information, and the **result** indicates that your call was successful (i.e., **ok**).

4 Using Functions

4.1 Understanding Function Categories

API functions are separated into categories, which are indicated by the first element after **/rest/** in the URL of the API function call. For example:

- all analysis-related API functions use the base URL: **/rest/analysis**
- all sample-related API functions start with: **/rest/sample**

4.2 Understanding Function Classes

All API functions can be separated into five different classes:

- **Get APIs** which return information related to items in the database.
- **Creation APIs** which create new entries in the database.
- **Update APIs** which update information for existing database items.
- **Deletion APIs** which delete elements in the database.
- **Miscellaneous APIs** which do not fall into any category above (e.g., sample submission).

API functions from each class all follow the same design principles. The next section explains generic aspects of the first four API function classes.

4.3 Get APIs

Get APIs are called by issuing an **HTTP GET** operation on the respective URL. Depending on the API call, they return either lists of datasets, or one dataset. For example, to get all analyses you call this URL:

GET /rest/analysis

This returns a list of all analyses datasets in the VMRay Platform database. To get only the dataset of a specific analysis with ID **123**, call:

GET /rest/analysis/123

This does not return a list but the dataset itself. This call fails if the dataset does not exist in the database.

4.3.1 Using Filters

Results can be filtered using various parameters that depend on the API function being called. For example, to show all analyses of the user with an ID of **1**, and a VM with an ID of **10**, call:

GET /rest/analysis?analysis_user_id=1&analysis_vm_id=10

For convenience, many filters can even be applied at the URL level. For example, to show all analyses of the user with an ID of 1, just call:

GET /rest/analysis/user/1

Available parameters and filters are listed in the *API References*.

Filters often require that you specify IDs as reference. In the above example, you can only filter for users by specifying the user ID of the user (and not the email address, etc.). This is an intentional design choice and so you must first retrieve the ID of the dataset being referenced. For example, to get the user ID of the user with the email **foo@bar.com**, call:

GET /rest/user/email/foo@bar.com

4.3.2 Understanding the Common Query Parameters

Get APIs usually support common query parameters as indicated in the *API References*. These common parameters are:

Parameter	Data Type	Description
_count	BOOL	Returns the number of datasets as an integer instead of actual data. Allows you to count jobs, analyses, etc.
_limit	INT	Limits the maximum number of returned datasets to this value. If result truncation is active, this number cannot be greater than the value in the System Settings.
_max_id	INT	If the API call returns data with an ID field, this parameter requires each item ID to be equal to, or less than, this number.

Parameter	Data Type	Description
_min_id	INT	If the API call returns data with an ID field, this parameter requires each item ID to be equal to, or greater than, this number.
_order	STR	If the API call returns data with an ID field, this parameter orders the results by the ID either **asc** or **desc**. The default ordering varies and depends on the called API function.

4.3.3 Circumventing Result Truncation

For complete information about result truncation, consult the *Understanding Result Truncation* section of the Understanding the Result Format chapter. The **_max_id** and **_min_id** parameters allow you to circumvent result truncation: When you perform an API call that returns truncated data and the data is in ascending order, you can take the last ID value returned and provide it in a second API call in the **_min_id** parameter. You can then access any items that were previously truncated. Use the same approach for descending data by using the **_max_id** parameter.

For example, if you get a truncated list of all analyses, ordered by analysis ID in descending order using **GET /rest/analysis**, you can take the smallest ID returned (we assume this is 1000 here) and do a second call:

GET /rest/analysis?_max_id=1000.

This returns only analyses with an analysis ID smaller than 1000 and thus yields data that was previously truncated.

4.3.4 Interval Searching

Some GET API parameters allow the use of intervals. This enables you to search for dataset parameters within a range. For example, you can query all analyses that were created within a certain period of time. The start and end range has to be separated by a tilde (~) character in the search. The start interval is included in the result, whereas the end interval is not.

For example, the following query returns all analyses created on 2021-05-27:

```
curl -H "Authorization: api_key xxJs8ca1lCJaKaUbwcneKP5jRUjLJ0TF" -X GET "https://
cloud.vmray.com/rest/analysis/created/2021-05-27T00:00:00~2021-05-28T00:00:00"
```

It is also possible to omit the start and end interval. The API then returns all datasets in which the searched-for value is greater than, or equal to, the search value, or all values in which the searched-for value is smaller than the search value.

For example, the following query returns all analyses with an Analysis Archive that is less than 1024 bytes:

```
curl -H 'Authorization: api_key 4CJxiTKEP3F5tDSGttPQ2FrBYQ67pJUe' -X GET 'https://localhost/rest/analysis/size/~1024'
```

Likewise, the following query returns all analyses that have an Analysis Archive greater than or equal to 1024 bytes:

```
curl -H 'Authorization: api_key 4CJxiTKEP3F5tDSGttPQ2FrBYQ67pJUe' -X GET 'https://localhost/rest/analysis/size/1024~'
```

The parameter information tables in the *API References* indicate for which parameters intervals can be used.

4.4 Creation APIs

New items can be created by issuing a **POST** operation on the appropriate URL. Contents of the respective data fields must be passed as parameters. For example, to create a new snapshot, you call:

POST /rest/snapshot?snapshot_name=test

The *API References* list which parameters must be passed, and which are optional, with default values. Note that you cannot break any consistency constraints in the database. For example, you cannot create a user with an email address that is already used by another user. This will yield an error.

This API call returns the newly created item.

4.5 Update APIs

Item fields can be updated by issuing a **POST** operation. You can only update one item with each request and the item must be referenced using its unique ID. For example, to update the Group with an ID of **123** to be **test,** call:

POST /rest/group/123?group_name=test

This API call returns the updated item.

4.6 Deletion APIs

Items can be deleted by issuing a **DELETE** call. You can only delete one item with each request and the item must be referenced using its unique ID. For example, to delete the Group with ID **123** from the database, call **DELETE /rest/group/123**.

5 Understanding Parameters

API functions can include required or optional parameters that can be passed as GET parameters in the URL. API calls that use POST requests can also pass parameters as form-encoded POST parameters. POST parameters are not restricted in length and should thus be used for long data. Consult the *API References* to see all of the parameters of each API call.

The following parameter types are supported:

Name	Description
BOOL	Boolean value, which can be either **true** or **false**.
DATE	Date string in ISO 8601 format. All date values are in Coordinated Universal Time (UTC).
FILE	Binary file using multipart/form-data according to RFC 7578, or a URL, as described below.
FLOAT	Floating point number.
INT	Signed integer value.
STR	String.
TIME	Time string in the format: **yyyy-mm-dd hour:minute:second.microsecond** (microsecond is optional). All time values are in UTC.

Integer parameters can be set to NULL by providing the special constant **null** instead of a value.

Please note that GET parameters are restricted in their length. By default, the entire GET parameter list should not surpass more than 32kb. Otherwise, the API call may fail with a corresponding error message.

5.1 File URL Support (On Premises only)

When uploading a binary file (such as during sample submission), it is possible to provide a reference to the sample file instead of the sample file itself. This enables you to submit samples without transferring the file over the API HTTP connection.

In large scale installations, where many sample submissions happen parallelly, this can greatly reduce the load on the Platform Server. Currently, Analyzer supports its own protocol called **vmray-file**, as described below.

Additionally, you can use the **http://** scheme to refer to files that must be fetched from a remote HTTP server. So, you do not have to upload the file directly, but instead, you can provide a reference to any HTTP location you want.

5.1.1 Referencing a Previously Submitted Sample

To reference a sample file that already exists in the database (e.g., for resubmitting an existing sample), you can use:

vmray-file:///sample/<sample_id>

For example, if you want to resubmit the **sample** with ID **123**, use the following URL:

vmray-file:///sample/123

The user that this API key is assigned to must have access to this sample file. That is, the user (or any of their API keys) must have submitted the sample file at least once previously.

You can use exactly the same mechanism described in the previous paragraph for Prescripts, using the URL:

vmray-file:///prescript/<prescript_id>

5.1.2 Using a Local Sample Upload Directory

As mentioned previously, in large scale On Premises installations, it can be beneficial not to submit sample and/or prescript files using HTTP because transferring sample files via HTTP introduces noticeable load on the Platform Server processes, which can, in turn, create a bottleneck.

To fix this problem, the VMRay Platform provides an alternative solution by using a local sample upload directory. This requires that your Platform Server already has the sample file available in its local file system (for example, by using an NFS mount point) and the submitter thus uses a reference to this file instead of uploading it.

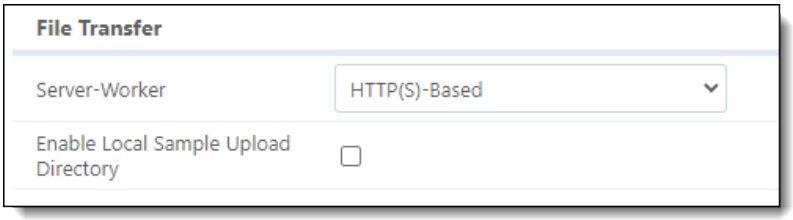
There are strict requirements for this feature:

- The sample file must be stored in the **/opt/vmray/var/sample_upload** folder or any of its sub-folders
- The feature must be enabled in the System Settings (it is disabled by default).

It is not possible to traverse out of this folder for obvious security reasons.

To turn this setting on:

1. Log in to the Console.
2. Click on **System Settings** in the Control panel.
3. Click the **GLOBAL CONFIGURATION** button.
4. Scroll down and locate the **File Transfer** section:



5. The second setting turns this feature on, so check it:

6. Scroll down to the bottom of the page and click the **SAVE** button.

When the sample upload folder is enabled, every Platform user in the system can access any file that is present in this directory or any of its sub-directories, if they know the file name. This means that when using multiple VMRay Platform users, user A can submit files from user B that are stored in the sample upload folder and which have not been deleted yet.

The VMRay Platform ensures that it is not possible to traverse out of the sample upload folder **/opt/vmray/var/sample_upload**, and it makes no effort to handle symbolic links to other folders inside that directory in a special way. Symbolic links are treated as normal files or directories. You have to make sure that you do not create any symbolic links to other locations in the system which can contain sensitive data.

6 Understanding the Result Format

Almost all API calls return JSON objects. To facilitate error handling and parsing, API return data is embedded into a generic structure in the form of a JSON dictionary that is used by all API calls that return JSON data. The dictionary has the following members:

Field	Description
result	String describing whether the API call was successful or not: **ok** if successful, otherwise **error**.
error_msg	If the result is **error**, this field contains a detailed error string describing the reason for the failure.
data	If the result is **ok**, this field contains the actual result data of the API call. The value depends on the API function that was called. In most cases, this is a list of the returned data sets.
continuation_id	If the result data is too large to be returned in one request, then this field contains an ID that can be used in subsequent calls to retrieve any remaining data (see Understanding Result Caching).
truncated	Boolean value indicating whether result data was truncated because it was huge (see Understanding Result Truncation).

The only API calls that do not return JSON objects are those that return files (for downloading analysis archive, sample files, etc.). In this case, the file data is returned directly unless an error occurred.

6.1 Understanding Result Caching

Many API calls return lists of data. For example, the API call **GET /rest/analysis** returns a list of all analyses in the system. Since these lists can be very large, the Platform may not return all data at once, so as to reduce network and server load. If this happens, a **continuation_id** field is provided in the results that holds an ID. This ID can be used in a second API call to retrieve any remaining result data. This continuation call can again contain a continuation ID (which can be different from the previous one). Hence, it might be necessary to perform multiple continuation calls to retrieve all data.

Any database changes applied in the meantime do not affect future continuation calls. For example, if you request a list of samples, then do a continuation call because the result was too large, then any new samples that were submitted in the meantime before the continuation call **will not be part of the result**.

Continuation results are stored for a certain amount of time (usually some minutes) on the Server until they expire. If they are not requested before expiry, the system automatically discards them.

Result caching can be configured (and disabled) in the System Settings.

6.2 Understanding Result Truncation

In addition to result caching, VMRay Platform can also perform result truncation. Result truncation constitutes a more vigorous approach to limit the results of an API call. If an API call yields more than X result items, the result list is **rigorously** truncated to that number. It is not possible to retrieve any of the truncated items directly, not even with result caching.

The reason result truncation is implemented is because with large databases that have many data sets, some API calls might slow down the system. Result truncation ensures that this cannot happen.

It is still possible to retrieve any truncated items indirectly by using the **_min_id and _max_id** parameters, explained in the previous chapter.

7 Looking at a Simple Example

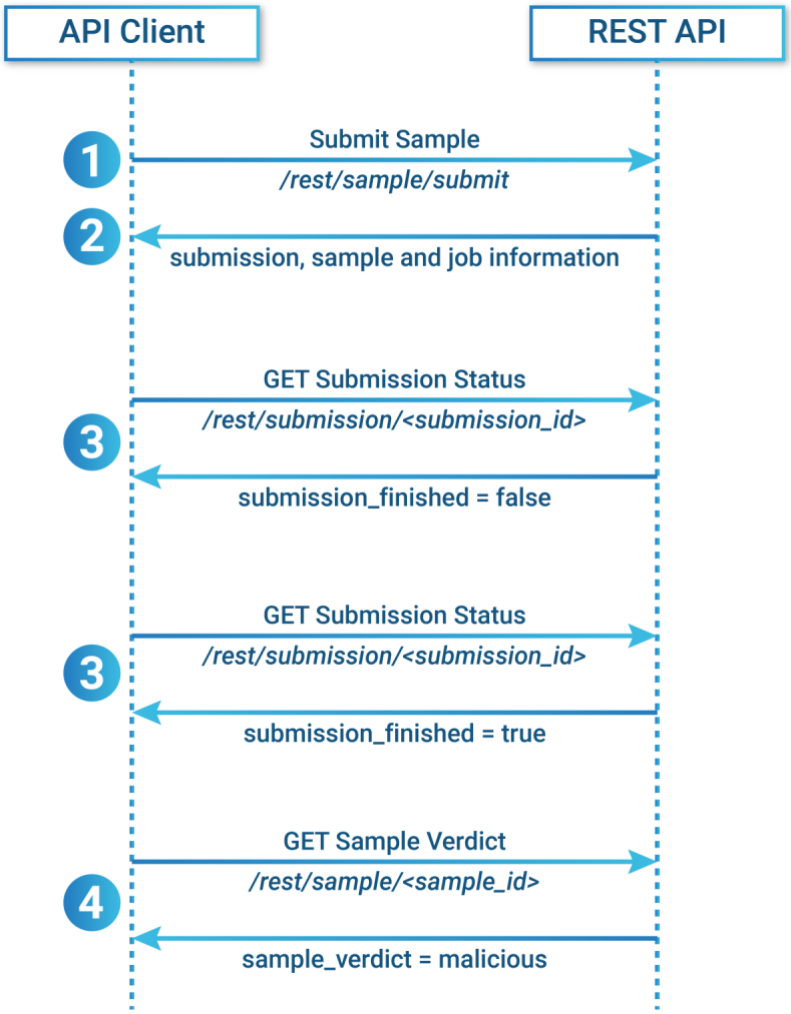
The most common use case for the API is to submit a sample and wait for all jobs to finish before retrieving the analysis results. The general workflow in this case is depicted on the next page and explained below:

1. Submit the sample using a POST to the **/rest/sample/submit** endpoint. Ensure that the data is posted using **Content-Type multipart/form-data** according to **RFC2388**. Curl and most HTTP libraries automatically do this for you.

2. The API returns a dictionary of several lists holding all the jobs that have been created. The type of jobs generated depend on the configured Jobrules. It can be VMRay Platform jobs, Reputation jobs, or VirusTotal jobs. Furthermore, the dictionary also contains a list of submissions that were created in the database. Normally, you only have one submission. Multiple submissions can only occur if you upload an archive and set the **compound_sample** parameter to **false**. In that case, all of the files in the archive are treated as if they were submitted separately and each file generates one submission. See the Tutorial: Submitting Archives (Recursive Analysis) for more information.

3. To check whether all jobs have finished, you can periodically query each submission using a GET to the **/rest/submission/<submission_id>** endpoint. This returns a dictionary and the key **submission_finished** is set to **true** as soon as all jobs for this submission have finished. As soon as the submission has finished, the **submission_has_errors** field indicates whether an error occurred during the analysis (i.e., at least one analysis finished with an error).

4. If you only want to know whether the VMRay Platform considers the file to be clean or malicious, then you should query the sample information at **/rest/sample/<submission_sample_id>** and read the **sample_verdict** field. It can be either **malicious, suspicious, clean,** or **unknown**. For the first submission, it requires the submission to be finished before a non-unknown sample verdict can be seen.



To query detailed individual analysis results after all jobs have finished, you can use a GET request to the following endpoints:

- **/rest/analysis/submission/<submission_id>**: Returns a list of all VMRay Platform analyses related to the submission.
- **/rest/reputation_analysis/submission/<submission_id>**: Returns a list of all Reputation Analyses related to the submission.
- **/rest/vt_analysis/submission/<submission_id>**: Returns a list of all VirusTotal analyses related to the submission.

The next several chapters provide step-by-step information on how to use the API.

8 Tutorial: Submitting Sample Files and URLs

This chapter provides step-by-step instructions for using the API, starting with the **system_info** endpoint. Next, we look at how to submit a sample and see the results. Almost every API endpoint corresponds to actions that you can perform in the Console, so this chapter also illustrates how to use the Console with the API.

8.1 Getting System Information with GET /info

To start out, let's revisit the endpoint that was introduced in earlier chapters: **/system_info**, and use it to verify that you have a valid API key and that you are using the correct Host Endpoint. Execute this call at a command prompt, but of course, substitute your own API key and the appropriate HTTPS address:

```
curl -H "Authorization: api_key zJs8ca1lCJaKaUbwcneKP5jRUjLJ0TF" -X
GET "https://eu.cloud.vmray.com/rest/system_info"
```

If you are not sure which Host Endpoint to use, or if you are not sure what your API key is, consult the Creating API Calls chapter.

The results of this call display as a JSON object with basic information about the system, such as the **version** of the Platform that you are using, which in this example is **4.3.0**:


```
{
  "data": {
    "api_items_per_request": 100,
    "file_param_http_scheme_enabled": true,
    "max_api_items": 1000,
    "version": "4.3.0",
    "version_major": 4,
    "version_minor": 3,
    "version_revision": 0,
    "webif_alias": "EU Cloud",
    "webif_base_url": "https://eu.cloud.vmray.com",
    "webif_max_sample_size": 262144000,
    "webif_max_upload_size": 262144000
  },
  "result": "ok"
}
```


Note that the **result** at the bottom is **ok**. This final parameter always indicates if your call was successful or not. If the call was not successful, the result is **error**.

webif is a commonly used abbreviation for the Console (which was previously known as the Web interface) - which is the VMRay platform's GUI. So, for example, **webif_base_url** refers to the Base URL used to access the Console.

8.2 Submitting a Sample with POST /sample/submit

Let's get a bit more sophisticated. The fundamental power of the VMRay Platform lies in its ability to analyze potentially malicious files and URLs. To start an analysis, you need to submit a file or URL. So, in this section, we examine how you submit a file for analysis. We start by submitting through the Console so you can see the complete process. We then submit the same file using the API.

The Submitting a URL with POST /sample/submit section describes how to submit a URL.

8.2.1 Using the Console to Submit a File

Follow these steps to submit an EXE file for analysis using the Console:

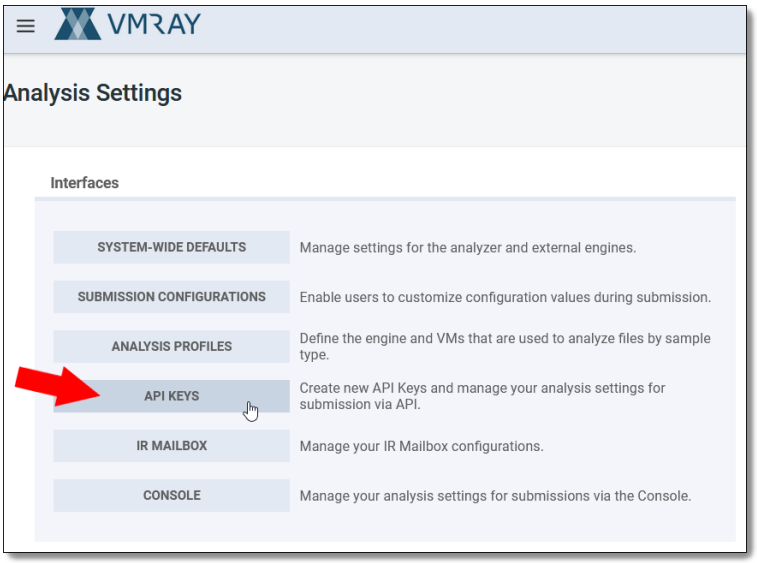
1. Log in to the Console.

   If you are not yet familiar with using the Console, consult the *Console Quick Start Guide*.

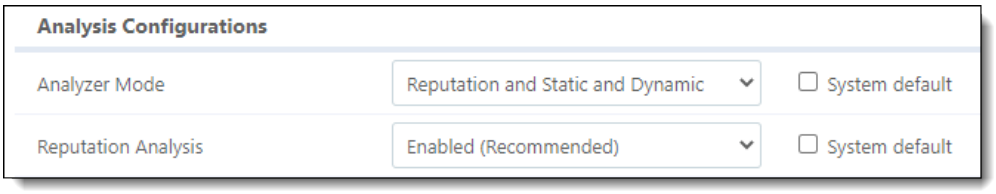
2. Let's first verify what types of analysis will be performed. Click the **Analysis** option in the Settings menu.

3. Click the **API KEYS** button:



4. Find the API key you set up earlier (in the Creating API Calls chapter). Click the **Edit** option in the **Actions** column:

5. The page that displays has the analysis settings related to your API key. Scroll down to the **Analysis Configurations** section and you see these two settings:

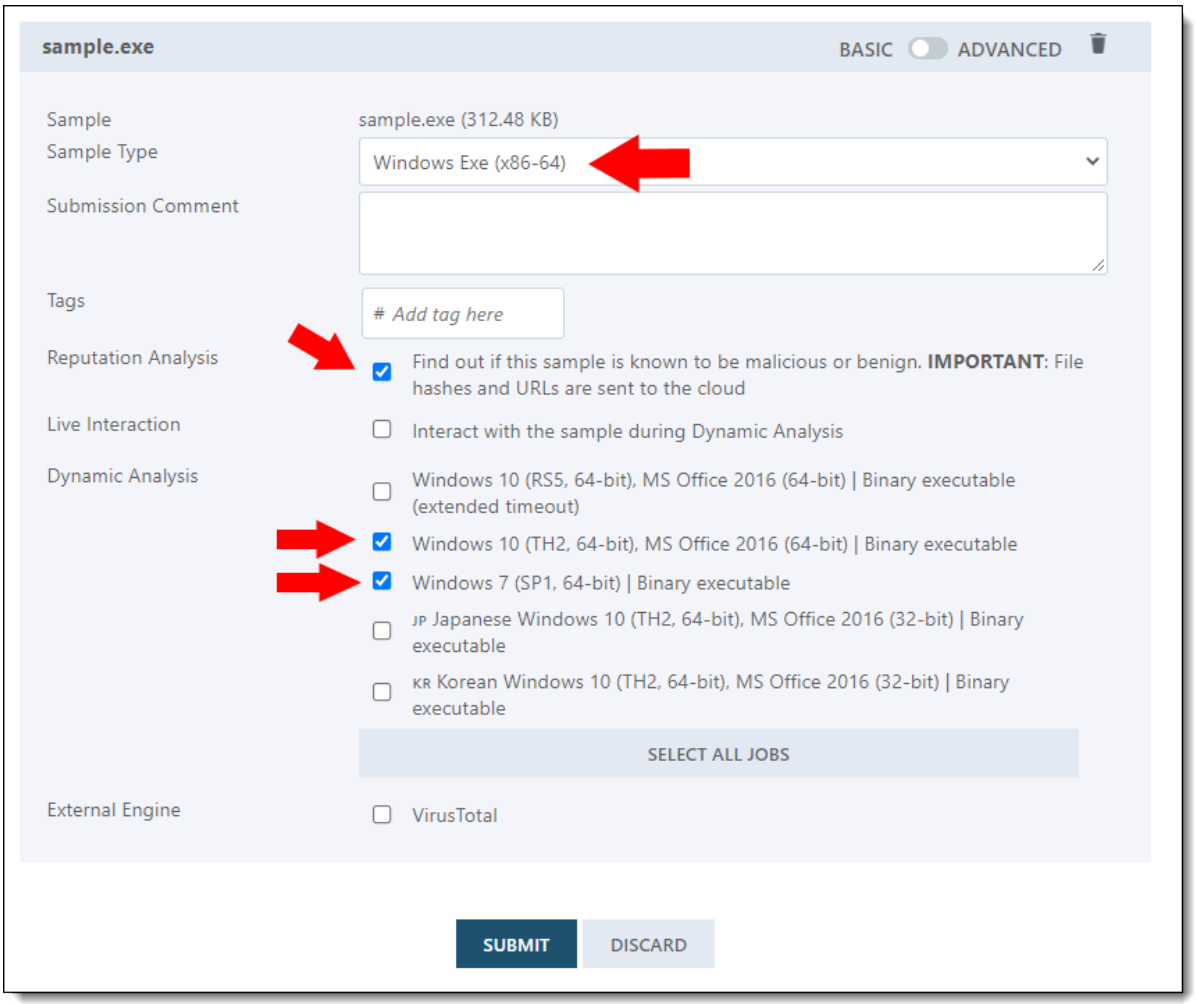


These settings together define how many of the three different types of analysis will be performed for all of your API submissions.

First, **Reputation Analysis** can be enabled or disabled. In this example it is **Enabled**, but it is possible it may be disabled for your company by your Account Manager (for Cloud customers) or your Administrator (for On Premises customers). If it is disabled for your company, your analysis results will not include Reputation Analyses and your screens may differ from the ones shown below.

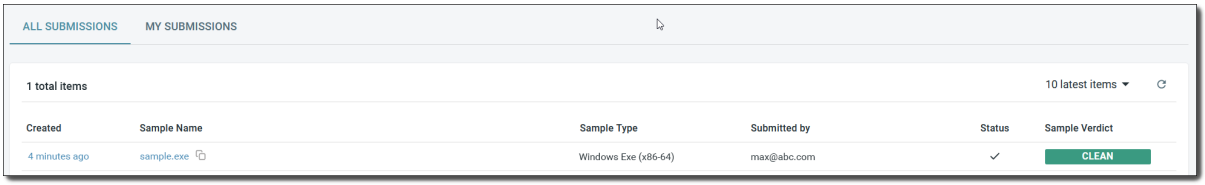
The **Analyzer Mode** defines which of the three types of analyses are performed, and in this example, all three are performed: **Reputation** Analysis, **Static** Analysis and **Dynamic** Analysis.

6. Submit a sample EXE file. There are three ways to submit a sample. Drop the file onto the **UPLOAD SAMPLE** box on the Dashboard, or click inside the UPLOAD SAMPLE box and a file explorer opens so that you can select a file, or click the **Upload Sample** button and drop the file into the box on the Sample Submission page (shown below). You then see the submission settings - most important among which is that the file is automatically recognized as a **Windows Exe (x86-64)** file, **Reputation Analysis** is turned on (you could turn it off at this point for this submission only), and by default, two Dynamic Analyses will be performed - in a **Windows 10** target environment and in a **Windows 7** target environment. Note that Dynamic Analysis environments may differ on your system as compared to what you see here based on how your Administrator or Account Manager has configured the Platform.

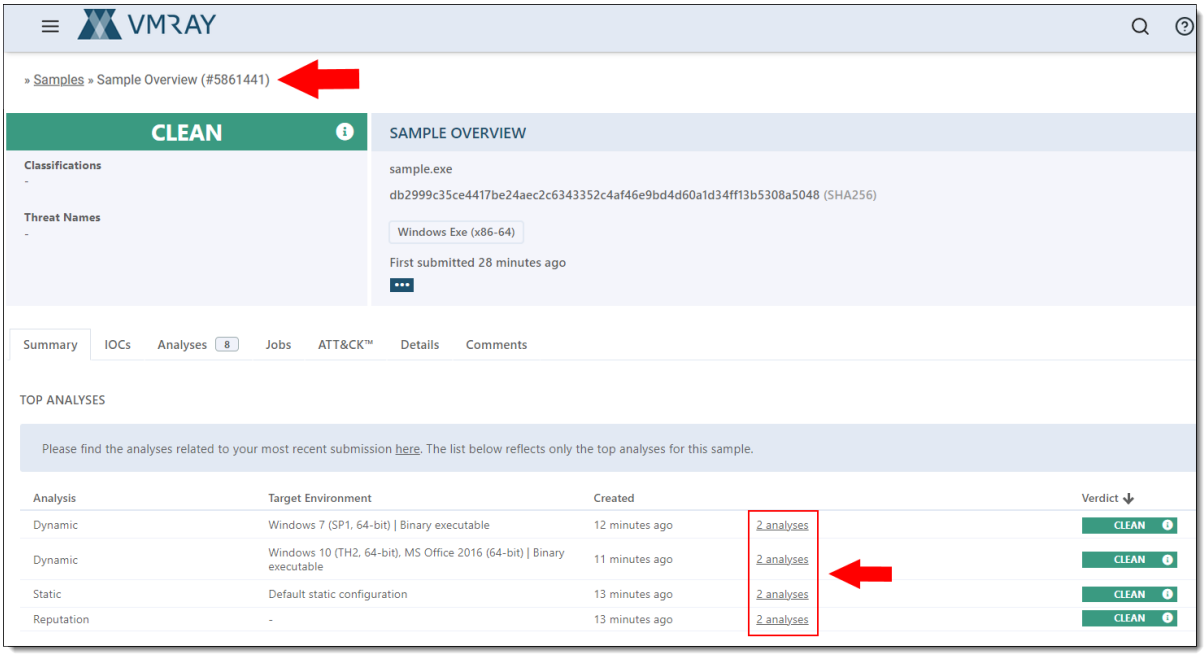


If you want to interact with a sample, turn **Live Interaction** on.

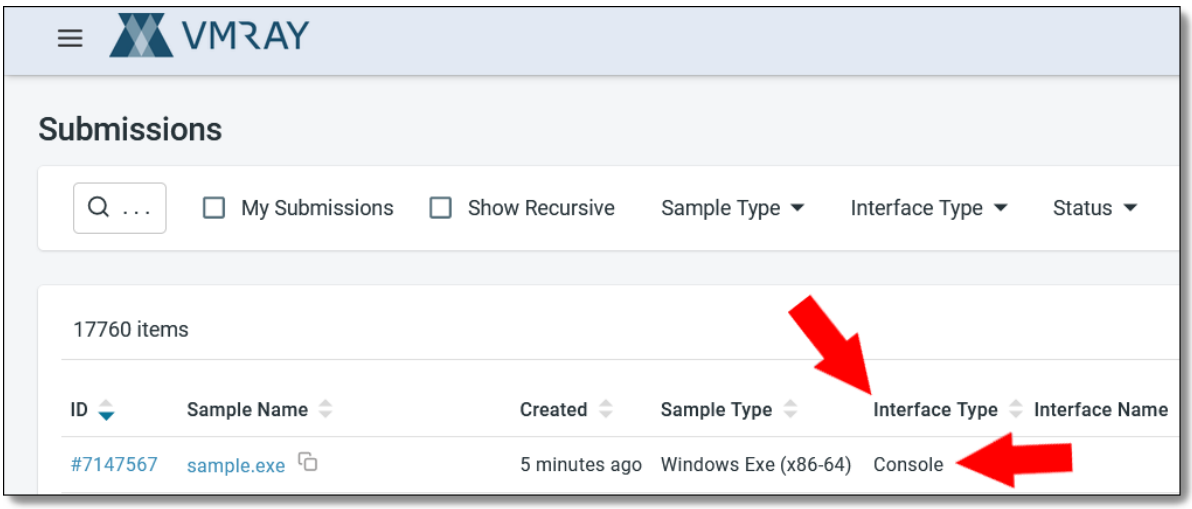
7. Click the **SUBMIT** button and you will be automatically redirected to the Dashboard where you can see your submission in the lower portion of the page:



8. Click anywhere in the row to see the **SAMPLE OVERVIEW** page which is your starting point for looking at all the analysis results, including the **Static** and **Reputation** Analysis reports, and the two **Dynamic** Analysis Reports:



9. Click the **SUBMISSIONS** option at the top of the window to display all submissions, and at the top of the list you see the **sample.exe** file:



Most importantly, take note of the **Interface Type** column. This indicates where a submission came from, and in this example, it came from the **Console**. If you submit using the API, your submissions will be shown here and the Interface Type will be listed as REST API.

8.2.2 Using an API Call to Submit a File

Submissions in Available Product Plans

- Note this inbound endpoint is not available in the DeepResponse product. DR users cannot submit samples via API.
- Depending on the licensed product plan, there might be restrictions when submitting certain sample types. Complete information is provided in the *Onboarding Guide - Appendix - Supported Product Features*.

Now, let's submit the exact same **sample.exe** file using the API. To do this, use a **POST** to the **/rest/sample/submit** endpoint and specify the file name in the **sample_file** parameter:

```
curl -X POST -H "Authorization: api_key zbFUXUEehHkL2no3CsFoiSf7qNZc1if" -F
sample_file=@sample.exe "https://eu.cloud.vmray.com/rest/sample/submit"
```

The resulting JSON displays in your terminal. It is too lengthy to show here so only an excerpt is shown. The key point is that various **jobs** process the submission to create the analysis results, and a variety of job information displays:

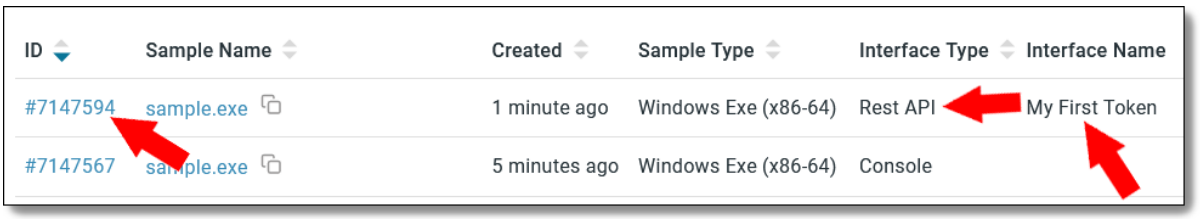
```
{
  "data": {
    "errors": [],
    "jobs": [
      {
        "job_created": "2021-05-27T11:42:40",
        "job_id": 7151696,
        "job_jobrule_sampletype": "Windows PE (x86-64)",

    "reputation_jobs": [
      {
        "reputation_job_created": "2021-05-27T11:42:40",
        "reputation_job_id": 561431,
    "static_jobs": [
      {
        "job_created": "2021-05-27T11:42:40",
    "submissions": [
      {
        "submission_analyzer_mode_analyzer_mode": "reputation_static_dynamic",
    ],
    "vt_jobs": [],
    "whois_jobs": []
  },
  "result": "ok"
```


Near the bottom, information related to the submission displays, such as the **submission_analyzer_mode_analyzer_mode**, which is set to **reputation_static_dynamic**. This corresponds to the Console setting we saw earlier where all three types of Analysis were turned on. Finally, at the bottom, you can see that the call was successful because the **result** is **ok.**

8.2.3 Looking at the API Submission in the Console

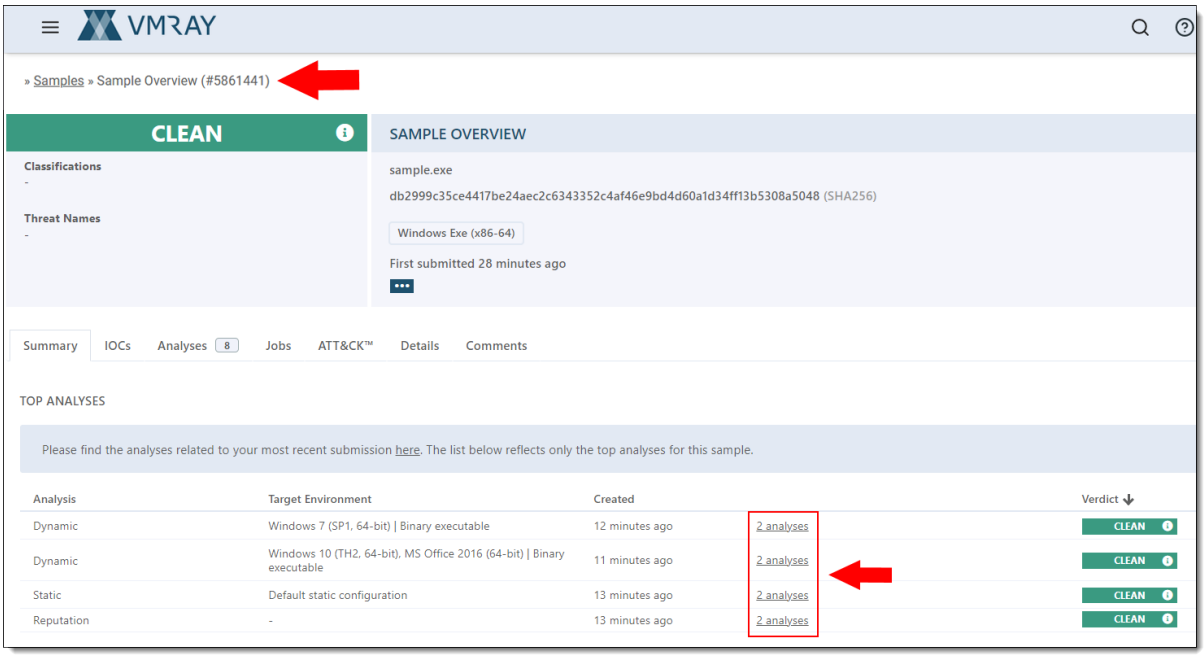
The Console is useful for tracking all of your submissions as well as the resulting analyses that are performed. To use the Console to verify the sample submission you just performed using the API:

1. Log in to the Console.
2. Click on **SUBMISSIONS** to see your API submission, which now displays just above the **Console** submission of the sample file, and the **Interface Type** column indicates that it came from the **Rest API**.



Also note that the **Interface Name** is **My First Token**. Recall that this is the name of the API key (sometimes referred to as a token) that we set up in an earlier chapter.
Take note of the Submission ID, which in this example is **7147594**. We use it in the next chapter, Tutorial: Getting Submission and Sample Data to get information related to this submission.

3. Now look at the **SAMPLE OVERVIEW** page. It is almost the same as the one we saw earlier, except that there are now 2 **analyses** listed for all four analyses. This is due to the fact that this file was already analyzed (earlier in this chapter):



Take note of the Sample ID, which in this example is **5861441**. We use it in the next chapter, Tutorial: Getting Submission and Sample Data, to retrieve information related to the sample.

8.3 Submitting a URL with POST /sample/submit

Submissions in Available Product Plans

- Note this inbound endpoint is not available in the DeepResponse product. DR users cannot submit samples via API.
- Depending on the licensed product plan, there might be restrictions when submitting certain sample types. Complete information is provided in the *Onboarding Guide - Appendix - Supported Product Features*.

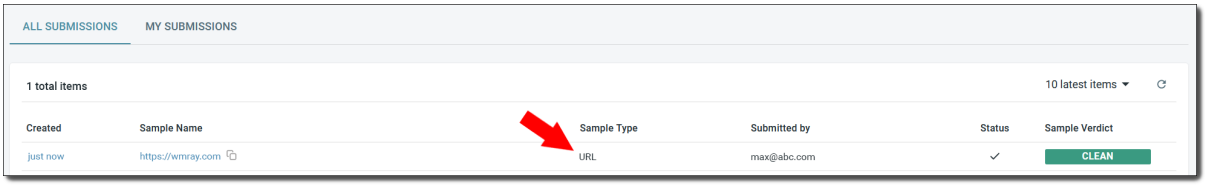
In addition to files, you can also submit URLs for analysis using the API.

1. Simply use the **sample_url** parameter as shown below. In this example, we analyze **vmray.com**:

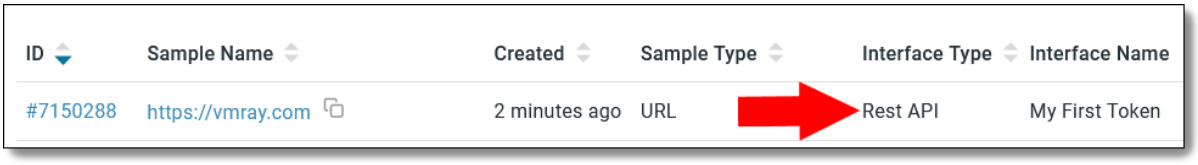
```
curl -X POST -H "Authorization: api_key zbFUXUEehHkL2no3CsFoiSf7qNZclif" -F
sample_url=https://vmray.com/ "https://eu.cloud.vmray.com/rest/sample/submit"
```

   The results display in the terminal and they are very similar to the file submission results so we need not take a look right now.

2. Instead, log in to the Console and you see that the analysis is already complete and the **Sample Type** is **URL**:



3. Click on **SUBMISSIONS** and you see the submission, which also came via the Rest API:



So submitting a URL is just as easy as submitting a file. In the next chapter, we look at how to retrieve the results using a variety of GET endpoints.

9 Tutorial: Getting Submission and Sample Data

9.1 Getting Submission Data with GET /submission

To get submission information for the sample file that you submitted in the previous chapter (Tutorial: Submitting Sample Files and URLs), execute a GET submission call with the Submission ID of **7147594** passed as the parameter at the end of the call:

```
curl -X GET -H "Authorization: api_key zbFUXUEehHkL2no3CsFoiSf7qNZclif" "https://
cloud.vmray.com/rest/submission/7147594"
```

Once again, the results are too long to list in full, so only some of the more important results are displayed in the excerpt below:


```
{
  "data": {
    "submission_analyzer_mode_analyzer_mode": "reputation_static_dynamic",
    "submission_created": "2021-05-27T11:42:40",
    "submission_filename": "sample.exe",
    "submission_finish_time": "2021-05-27T11:43:09",
    "submission_id": 7147594,
    "submission_interface_name": "My First Token",
    "submission_sample_id": 5861441,
    "submission_status": "success",
    "submission_user_account_name": "Max Mustermann",
    "submission_user_email": "max@abc.com",
    "submission_verdict": "clean",
    "submission_verdict_reason_description": "Known to be clean.",
    "submission_webif_url": "https://cloud.vmray.com/user/sample/view?id=5861441",
    "submission_whois_mode": "disabled"
  },
  "result": "ok"
}
```


These results are of particular importance:

- the **submission_verdict**, which is **clean**, as we saw in the Console
- the URL to open the Sample Overview page in the Console is in the **submission_webif_url** parameter
- the **submission_analyzer_mode_analyzer_mode** which displays the type of analysis used during the sample analysis; to query the string, use the curl command below:

```
curl --location 'https://{SERVER_NAME}/rest/sample/submit' \
            --header 'Authorization: api_key {VMRAY_API_KEY} \
             --form 'sample_file=@"/Users/path/sample.exe"' \
             --form 'analyzer_mode="reputation_static"'
```

Several other submission endpoints are available:

- GET /rest/submission
- GET /rest/submission/created/<submission_created>
- GET /rest/submission/finish_time/<submission_finish_time>
- GET /rest/submission/prescript/<submission_prescript_id>
- GET /rest/submission/priority/<submission_priority>
- GET /rest/submission/sample/<submission_sample_id>
- GET /rest/submission/type/<submission_type>
- GET /rest/submission/user/<submission_user_id>

The first endpoint retrieves all submissions, whereas the others let you retrieve subsets based on parameters you specify. For API users, the **submission_type** endpoint can be especially useful because it allows you to retrieve only one type of submission based on the interface used to submit, for example, you can:

- retrieve only API submissions using the endpoint: **/rest/submission/type/api**
- retrieve only Console submissions using the endpoint: **/rest/submission/type/webif**

The *API Reference* has complete information about all of these endpoints.

9.2 Getting Sample Information with GET /sample

The **GET /sample** endpoint enables you to get all of the metadata for the sample.exe sample file that you submitted, including signatures. Call it by adding the Sample ID of **5861441** to the end of the endpoint:

```
curl -s -k -X GET -H "Authorization: api_key zbFUXUEehHkL2no3CsFoiSf7qNZclif" "https://
cloud.vmray.com/rest/sample/5861441"
```

Some of the results are shown below to illustrate the level of detail that is available:


```
{
  "data": {
    "sample_filename": "sample.exe",
    "sample_pe_signature": {
      "signer": {
        "algorithm": "sha256_rsa",
        "issuer": {
          "common_name": "DigiCert SHA2 Assured ID Code Signing CA",
          "country_name": "US",
          "organization_name": "DigiCert Inc",
          "organizational_unit_name": "www.digicert.com"
        },
        "parent_certificate": {
          "algorithm": "sha256_rsa",
          "issuer": {
            "common_name": "DigiCert Assured ID Root CA",
            "country_name": "US",
            "organization_name": "DigiCert Inc",
            "organizational_unit_name": "www.digicert.com"
          "thumbprint": "92 C1 58 8E 85 AF 22 01 CE 79 15 E8 53 8B 49 2F 60 5B 80 C6",
          "valid_from": "2013-10-22T12:00:00+00:00",
          "valid_until": "2028-10-22T12:00:00+00:00"
        },
        "serial_number": "06 2E D3 29 B7 4E 5F 14 19 97 F8 FA 21 41 B7 3F",
        "status_checking": {
          "crl_urls": [
            "http://crl3.digicert.com/sha2-assured-cs-g1.crl",
            "http://crl4.digicert.com/sha2-assured-cs-g1.crl"
          ],
          "issuer_hashes": {
            "name": "c860f1fde14eca11366f0405545bc467037883c7984655be9ff0d2f6b3650021",
            "public_key": "e8c0b73c17bd6101a6adb82918d436855a14cdd824ae80b8e3cf2f8fbcff75af"

    "sample_type": "Windows Exe (x86-64)",
    "sample_verdict": "clean",
    "sample_verdict_reason_description": "Known to be clean.",
    "sample_vti_score": 0,
    "sample_webif_url": "https://cloud.vmray.com/user/sample/view?id=5861441"
  },
```

```
  "result": "ok"
}
```

The other **GET /sample** endpoints are listed below:

- GET /rest/sample
- GET /rest/sample/<sample_id>
- GET /rest/sample/created/<sample_created>
- GET /rest/sample/filesize/<sample_filesize>
- GET /rest/sample/md5/<sample_md5hash>
- GET /rest/sample/sha1/<sample_sha1hash>
- GET /rest/sample/sha256/<sample_sha256hash>
- GET /rest/sample/type/<sample_type>

The *API Reference* describes each of these in detail, and sample curl calls are provided for each.

10 Tutorial: Submitting Archives (Recursive Analysis)

Up until now, we have covered submitting files and URLs for analysis and getting the results of the analysis. Now we will return to the submission of files and URLs in order to look at Recursive Analysis.

10.1 Understanding Recursive Analysis

VMRay Platform products enable you to submit and analyze individual files and URLs. Quite often, though, files contain URLs or other files, and URLs tend to invoke other URLs or download files. To ensure comprehensive protection, we always analyze files and URLs within other files and URLs. This is known as Recursive Analysis, and it is invoked for:

- **Email files** - a file that has attached files or URL hyperlinks within the body text of the email.
- **Archive files** - a file that is comprised of one or more files that are compressed to form a single file, such as a ZIP file or a TAR file. Archive files always invoke recursive analysis.
- **URL** - a URL that invokes other URLs or downloads files.
- **HTML** - text-only files analyzed using the Web Engine.
- **macOS DMGs** - Disk Image (DMG) archives analyzed using the Static Engine.
- **Microsoft OneNote Documents** - .one files analyzed using the Static Engine.

Recommendation

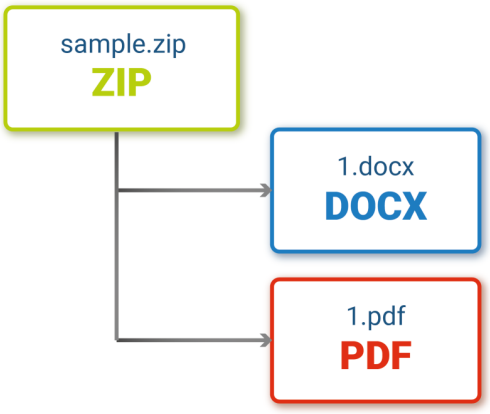
Since the Recursive Analysis goes to all levels of the analyzed sample, we recommend using it with **Analysis Caching** in "**Smart**" mode.

Analysis Caching optimizes the performance when resubmitting a sample. When set to Smart, it automatically determines whether caching should be used, considering a cache life span depending on the analysis type. This ensures an optimum balance between efficacy and performance.

You will find this setting in **Console** > **Settings** > **Analysis** > **System Wide Defaults**.

10.2 Looking at a ZIP File

To see recursive analysis in action, we'll look at a **sample.zip** file with a **1.docx** file and a **1.pdf** file within it, as depicted below:



We will also look at quota as we do this because there is increased quota usage when recursive analysis is invoked.

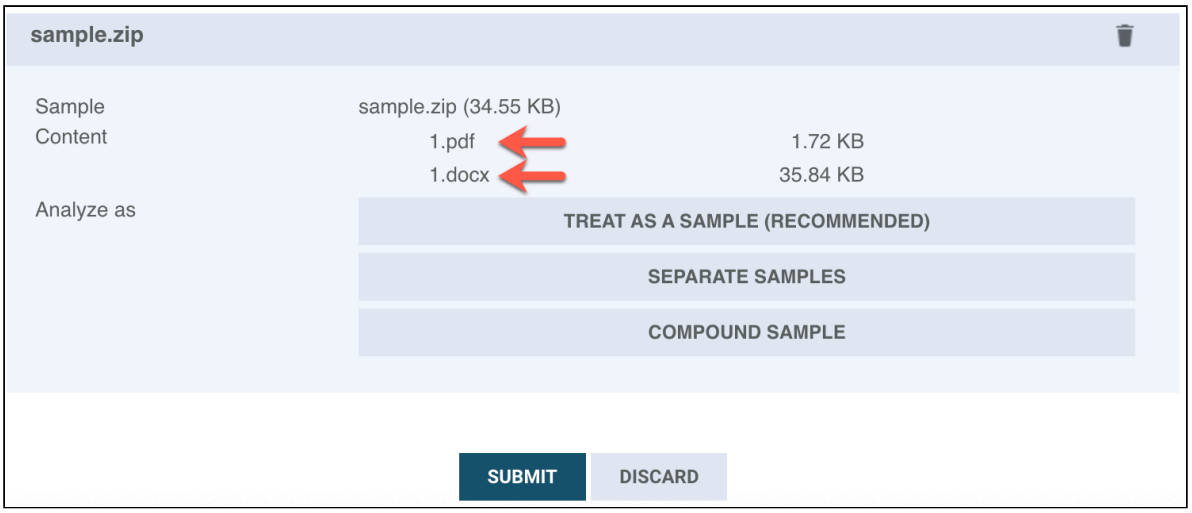
10.3 Submitting a ZIP Archive File using the Console

Let's look at submitting a ZIP file in the Console in order to understand some basic concepts that will be needed when we submit a ZIP file using the API:

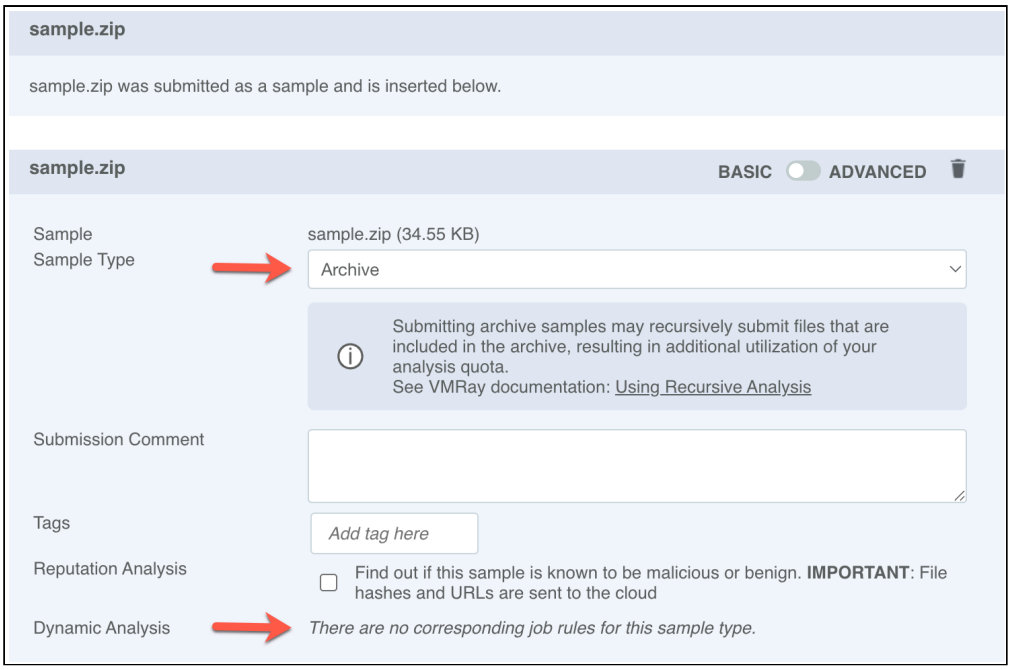
1. Log in to the Console and note the **Report/Verdict Quota Usage** on your Dashboard. You can follow along on your own system, but on our system, we have used 6 out of 100:

Reports Usage	6/100

2. Drop a **sample.zip** for analysis, and you can see that it has two files within it:



3. Select the first button: **TREAT AS A SAMPLE (RECOMMENDED)**. This indicates that you want to submit this file as an Archive and that recursive analysis will be performed if needed, and you now see:



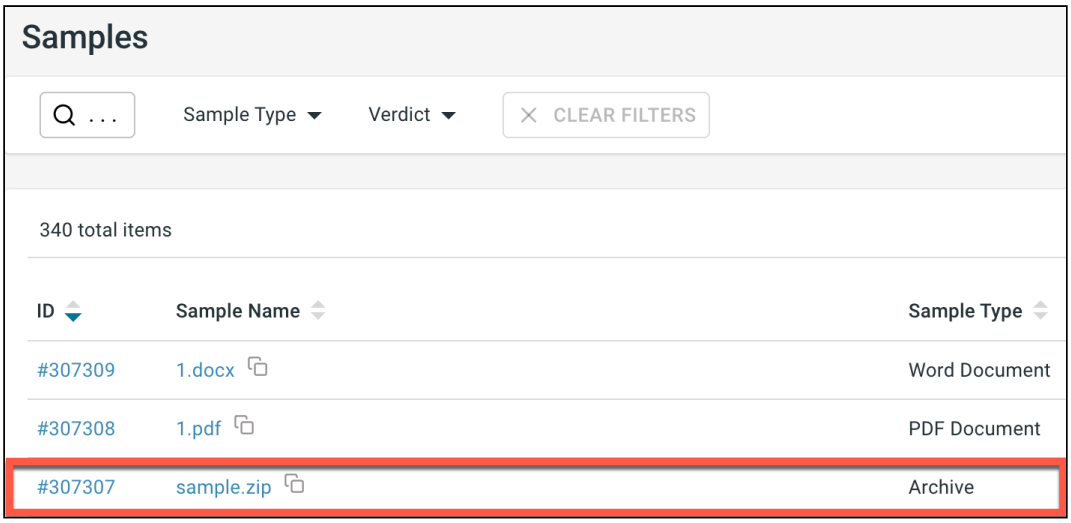
No **Dynamic Analysis** environments are specified here because this can only be determined during recursive analysis when the sample type of both files is identified.

4. Click **SUBMIT**.

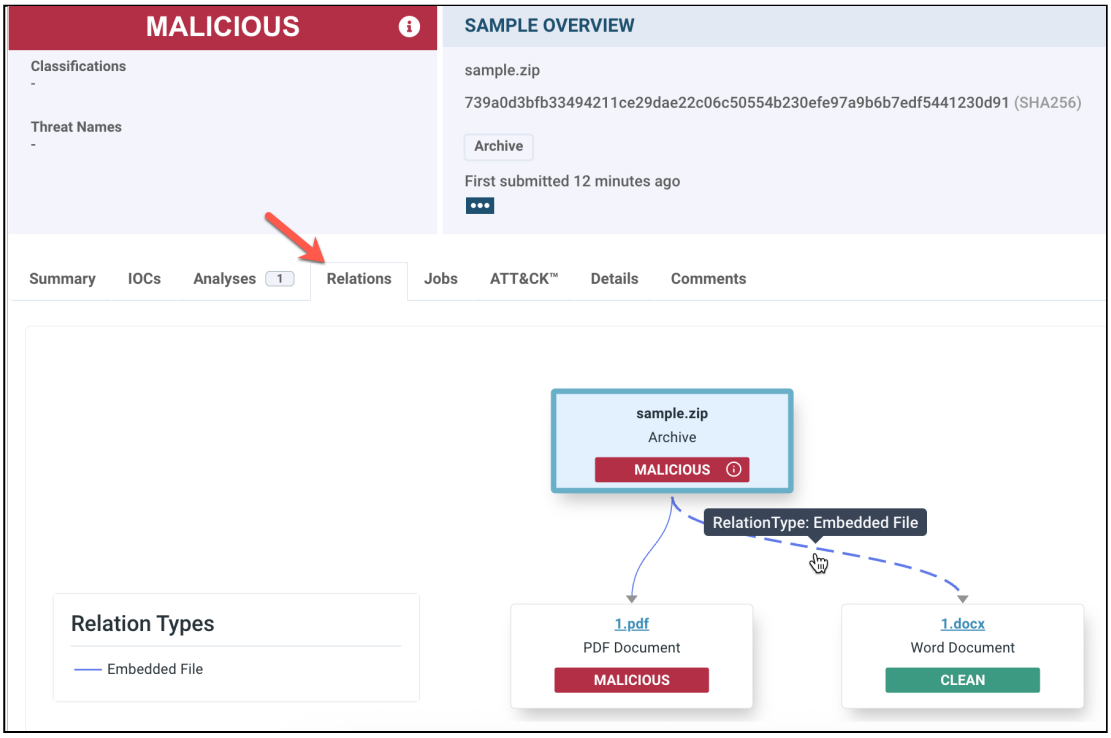
5. Return to the Dashboard and note the new **Reports/Verdicts Quota Usage**, which indicates 17 quota items were used (there were only 6 previously):

That's a big jump for just a single file. That's because of recursive analysis, which automatically decompresses the archive file and analyzes the individual files within it.

6. From the left menu, open **SAMPLES** and identify your submitted sample. In this case, 3 samples exist: the parent sample **sample.zip** and the samples contained within the archive, **1.docx** and **1.pdf**. Let's look at the **sample.zip,** which is the original archive file that was submitted.



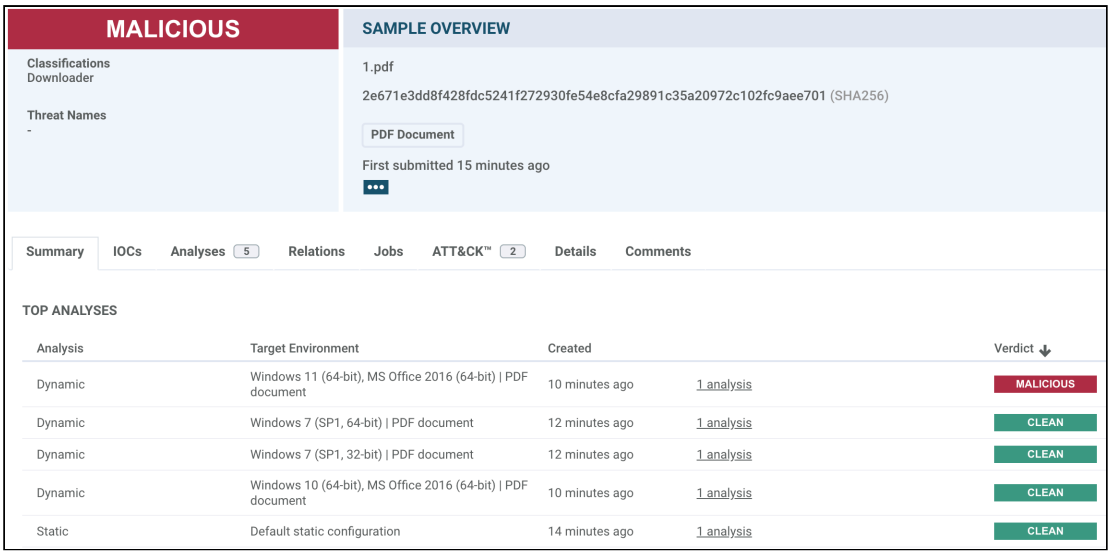
7. Select your sample and then click the **Relations** tab:



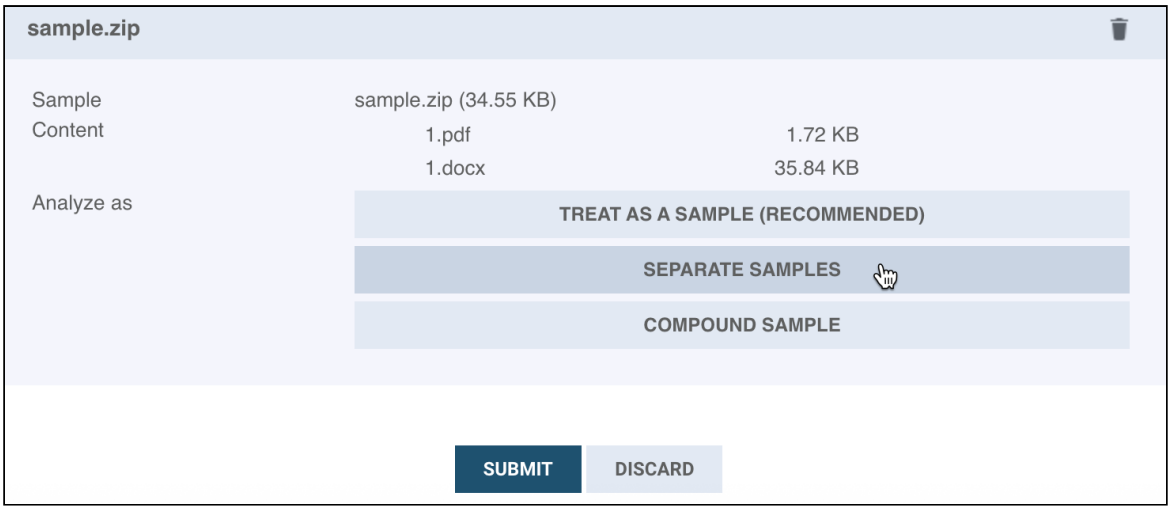
You can see that the **1.docx** and the **1.pdf** files have been identified as **Embedded Files**, and the **Relation Type** indicates that these two files were discovered within the ZIP and recursively submitted.

8. Click on the **1.pdf** tile, and you see that it is considered a standalone sample, and so it has its own **SAMPLE OVERVIEW** page. Scroll down, and you can see that several **Dynamic**

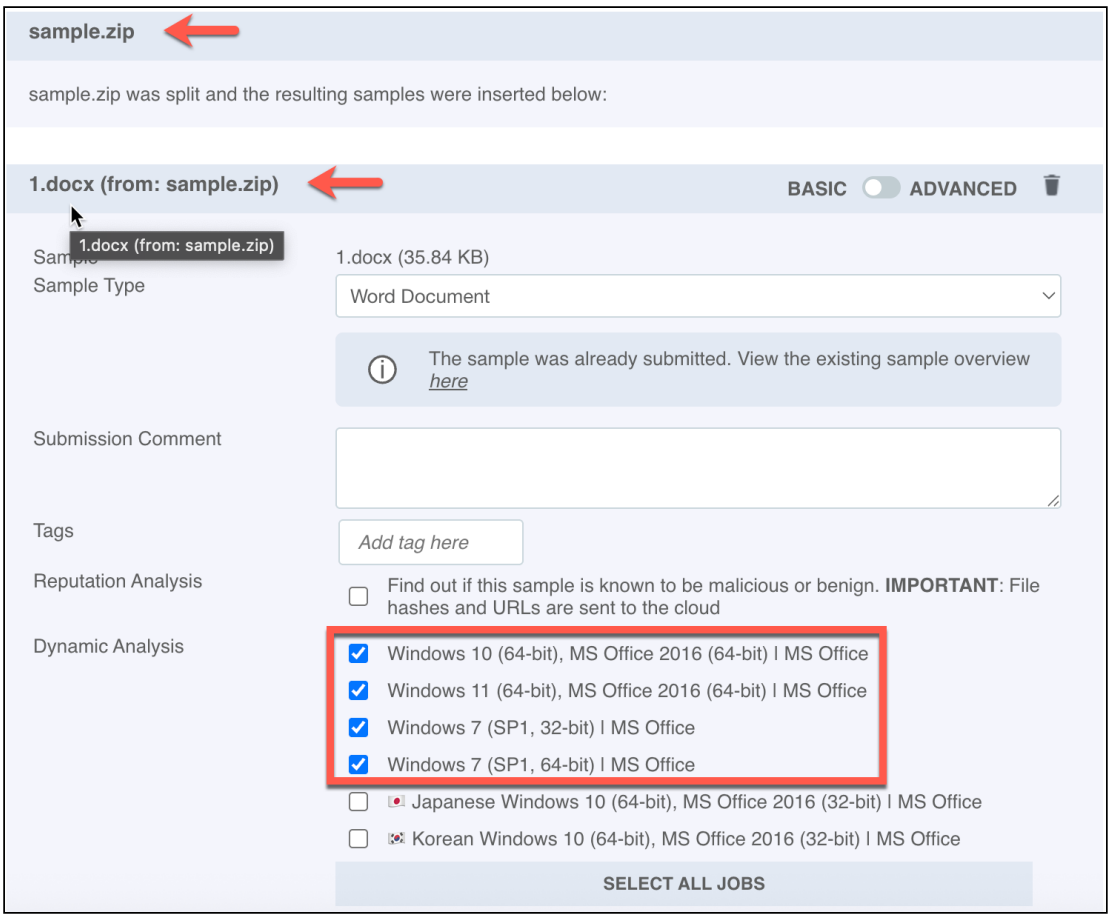
Analysis environments were used during the analysis of the **1.pdf** file, and this is why so much quota was used:



9. Now, let's drop the **sample.zip** for submission again, but this time choose **SEPARATE SAMPLES**:



10. This splits the **sample.zip** up and allows you to verify the **Dynamic Analysis** environments for each individual file, and you can see that this is where all of the Dynamic Analysis target environments for the **1.docx** are selected by default:

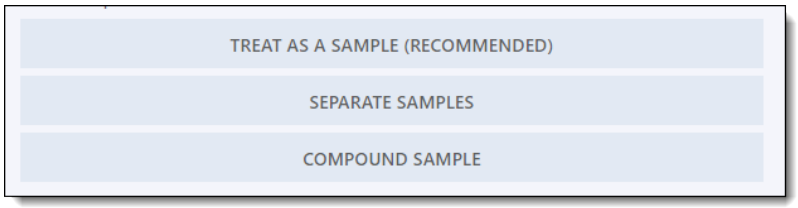


The **1.docx** also has a large number of target environments specified, which add up to the total quota usage.

11. There is no need to submit now, so click **DISCARD**.

10.4 Understanding the Three Archive Actions

The **archive_action** parameter allows you to specify one of the three actions you saw earlier in the Console, which is also a part of an API submission of an archive file:



The three corresponding parameter values for **archive_action**, respectively, are:

- **sample**: the archive is treated as a sample. This is the recommended action and we will use it below for the API submission. This is the option we used earlier in this section.

- **separate_samples**: every file in the archive is treated as a separate submission.
- **compound_sample**: the archive is treated as a compound sample with an automatically selected entry point (the **entry_point** allows you to override which file in the archive is the main file that should be executed).

The second and third values are covered in the *API References*, so we'll show only the first option in the next section.

10.5 Submitting an Archive with POST /sample/submit

Note this inbound endpoint is not available in the DeepResponse product - the DeepResponse user will not be able to submit the samples via API.

Use this call to submit the **sample.zip** for analysis and ensure that **archive_action** is set to **sample**:

```
curl -X POST -H "Authorization: api_key KObFUXUEehHkL2no3CsFoiSf7qNZc1if" -F
sample_file=@sample.zip -F archive_action=sample "https://eu.cloud.vmray.com/rest/sample/
submit"
```

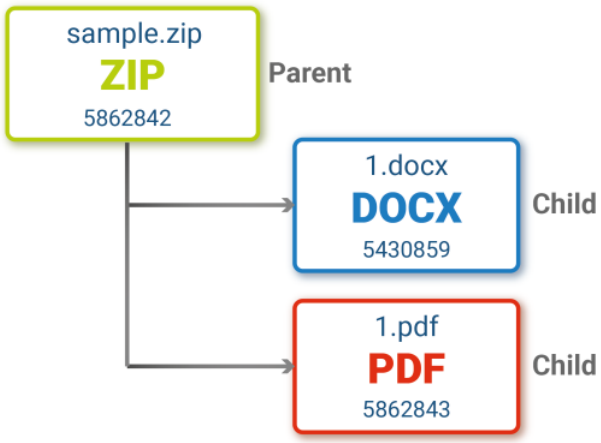
Review the resulting JSON and focus on this section:

```
"samples": [
  {
    "sample_child_sample_ids": [
      5430859,
      5862843
    ],
    ...
    "sample_id": 5862842,
```

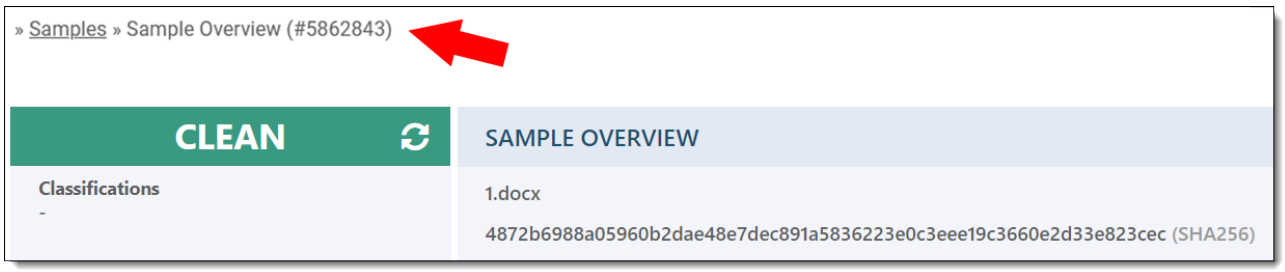
The Sample Identifier numbers are:

- **5430859** is the Sample ID of the child file: **1.pdf**
- **5862843** is the Sample ID of the child file: **1.docx**
- **5862842** is the Sample ID of the parent file: **sample.zip**

The parent/child relationships are depicted below:



You can confirm this in the Console by looking at the top of each SAMPLE OVERVIEW page, where the Sample ID is listed, as in this example of the **1.docx**:



Recall that the Quota Usage was previously 17, and it is now at 28, which means that additional quota items were used - the exact same number as when we submitted this file in the Console.

That's the basics of recursive analysis. Next, we'll look at analysis caching.

11 Tutorial: Understanding Verdict and Report Quotas

This chapter introduces our pricing model to illustrate how Verdict and Report Quotas are used, and how they impact your API results.

Prior to v4.3, Verdict Quota was generally referred to as our Detector product.

11.1 Understanding the Pricing Plans

With 2023.2.0 release, the pricing plans have been refined and consolidated. VMRay now has functionality and volume level separation in the product plans which can help organizations to select the features and capabilities that best meet their needs, while also providing the flexibility to scale their use of the product as their needs change.

- Product plans are defined by the selected product and supported feature categories. The plans for DeepResponse, FinalVerdict, and TotalInsight vary because each product has different capabilities.
- Product plans are defined by the number of total users, Analyses per month, features distribution, and other options.
- You can supplement your product with packs of additional users, extended support and connectors setup. Connectors are third-party cybersecurity solutions that can be integrated into our products built on the VMRay Platform. There are over 20 connectors available including EDR, SOAR and many more.
- Our products offer tailored price carriers and with this, they offer more stable approach to the conducted analyses quota.

Contact your VMRay representative if you would like to modify your product plan.

11.2 Viewing your Quota in the Console

Now let's look at how you can track quota usage in the Console. Quota stands for a prepaid number of Verdicts, Reports or users included in your licensed **VMRay product plan**.

11.2.1 Cloud

With the Cloud version, Account Managers can see more information related to quota so we look at that first, and then we look at what Standard Users see.

11.2.1.1 Account Managers

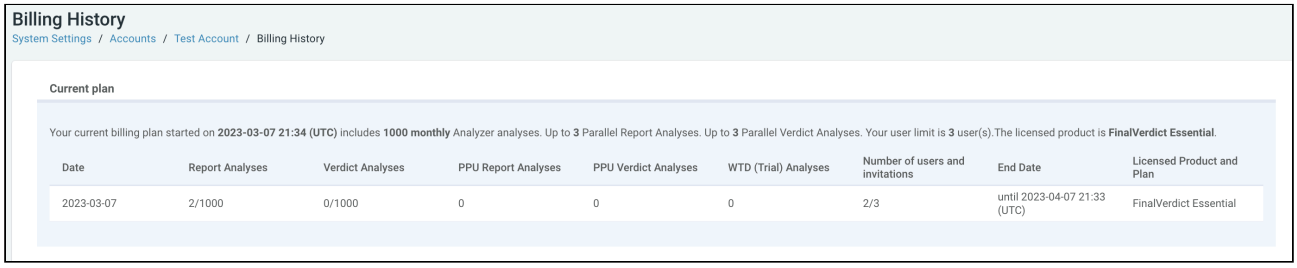
Cloud Account Managers can click on the Account Settings option in the Control Panel to see the current Report and Verdict Quotas.

Quotas	DeepResponse: 100 Reports per month | 10 users

- For FinalVerdict - only Verdict quota is shown
- For DeepResponse and TotalInsight - only Report quota is shown
- PPU information is presented only to our legacy users

Quotas	Analyzer (pre 2023): 100 Reports per month | 100 Verdicts per month | No PPU | 10 users

If you hover your mouse cursor over the triple dots on the far right, you are presented with the **View History** menu option. Click it and this page displays:

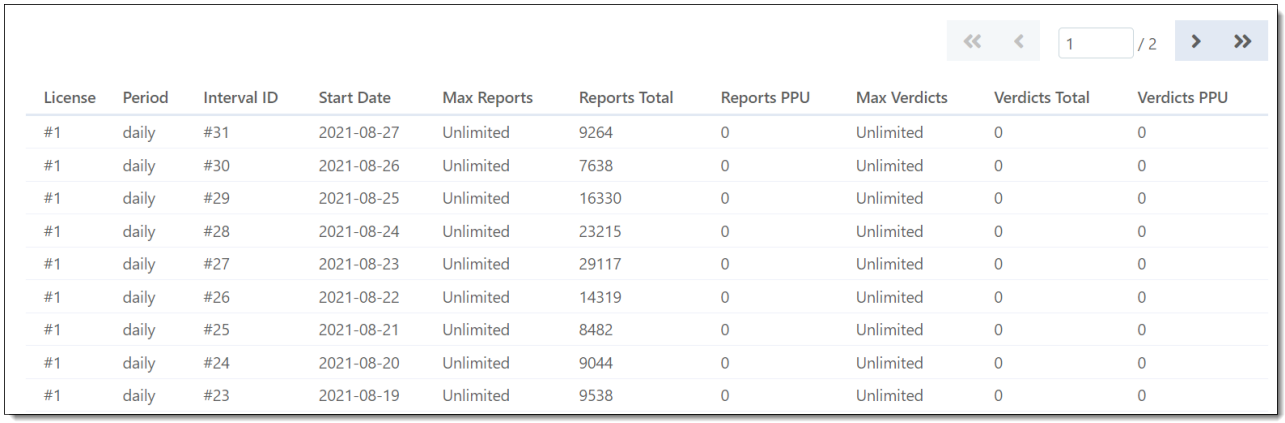


11.2.1.2 Administrators

Administrators can see their daily **Report Quota** and/or **Verdict Quota**, depending by the licensed product plan, by clicking on **System Settings** and then the **LICENSE** button:

Quotas	
Report Quota	1000
Verdict Quota	3000

In addition, more detailed information is available on the Quota History page, which you can access by clicking **System Settings > QUOTA HISTORY**:



License	Period	Interval ID	Start Date	Max Reports	Reports Total	Reports PPU	Max Verdicts	Verdicts Total	Verdicts PPU
#1	daily	#31	2021-08-27	Unlimited	9264	0	Unlimited	0	0
#1	daily	#30	2021-08-26	Unlimited	7638	0	Unlimited	0	0
#1	daily	#29	2021-08-25	Unlimited	16330	0	Unlimited	0	0
#1	daily	#28	2021-08-24	Unlimited	23215	0	Unlimited	0	0
#1	daily	#27	2021-08-23	Unlimited	29117	0	Unlimited	0	0
#1	daily	#26	2021-08-22	Unlimited	14319	0	Unlimited	0	0
#1	daily	#25	2021-08-21	Unlimited	8482	0	Unlimited	0	0
#1	daily	#24	2021-08-20	Unlimited	9044	0	Unlimited	0	0
#1	daily	#23	2021-08-19	Unlimited	9538	0	Unlimited	0	0

11.2.1.3 Standard Users

Standard Users can see their current usage on the Main Dashboard:

Account	Products Team
Product Plan	DeepResponse Unlimited
Reports Usage	0 / Unlimited
Account Reputation Setting	Disabled by Default ⚠

Depending on the licensed product, the Quota Usage is displayed differently:

- DeepResponse users will see only Report Usage
- TotalInsight users will see only Report Usage
- FinalVerdict users will see only Verdict Usage
- Analyzer legacy customers (pre-2023.2.0 release) will see Report and Verdict Usage

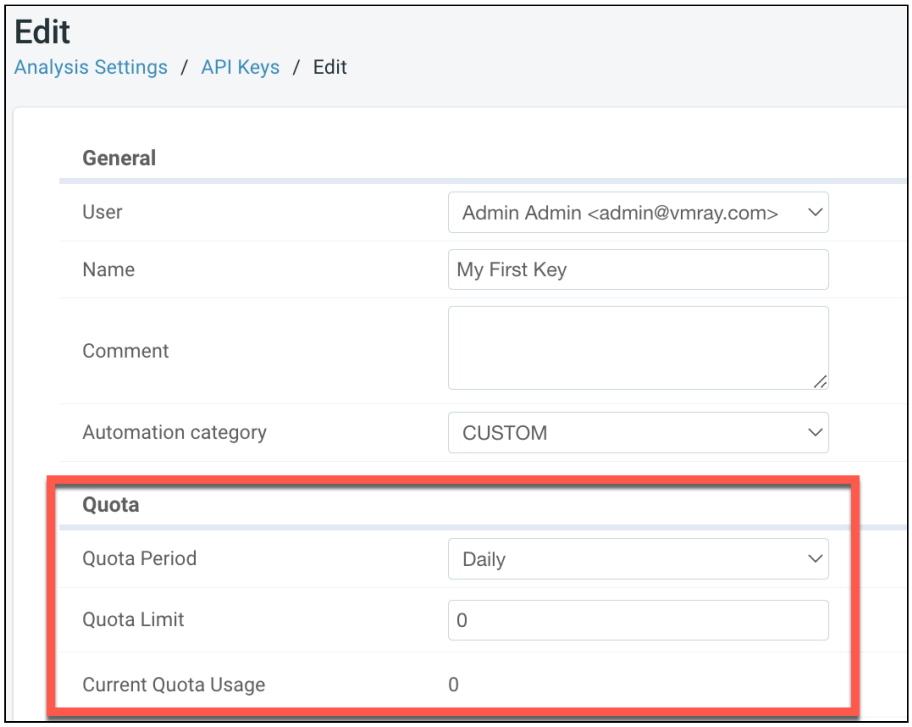
11.3 Checking Quota Usage per Submission

You can also check the quota usage using our REST API endpoint. In order to check the consumed quota **related to the queried submission** (including recursively triggered submissions), you can use the `GET/rest/submission` endpoint which returns the **submission_consumed_quota** attribute. You can read more about the specific endpoints in the *Cloud / On Premises API Reference* > *User Functions* > *Submissions and Prescripts*.

Note that the usage of REST API endpoints is limited to specific product and plans.

11.4 Defining Quota for API Keys

If you are an Account Manager or an Administrator, each API key that you assign to a user can indicate a **Quota Period**, **Quota Limit**, and you can see what their **Current Quota Usage** is:



*Pre-2023.2.0 release customers will also see Quota Type - allowing to specify the quota type to be used for this API key. For post-2023.2.0 customers, this setting is predefined by purchased product and plan.

12 Tutorial: Understanding Analysis Caching

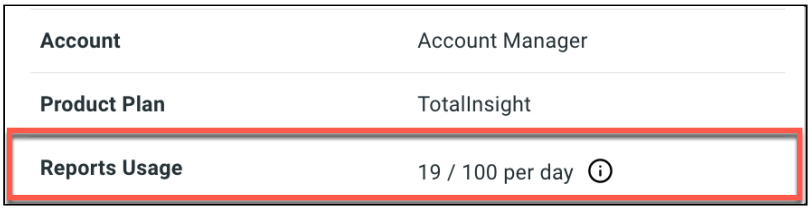
The REST API is particularly useful for building automated processes for analyzing samples. However, you should always monitor your quota usage as you build these processes, and use Analysis Caching to minimize unnecessary quota usage.

For example, let's say an EXE is submitted 10 times using the API, and two Dynamic Analysis target environments are defined for EXEs. Without Analysis Caching turned on, this would use 20 quota items. By turning analysis caching on, only 2 quota items would be used, since the cached analysis results would be used after the first time. This chapter walks you through an example to explain Analysis Caching in detail.

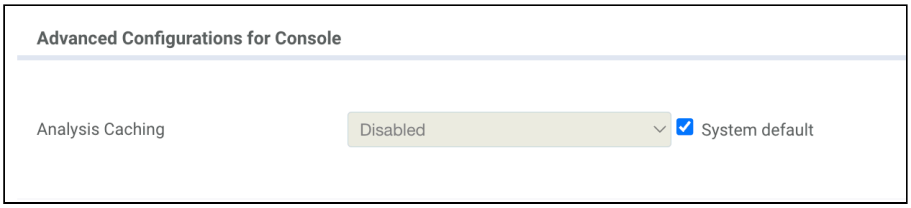
12.1 Submitting with Analysis Caching Disabled

Let's start by submitting a sample using the Console with Analysis Caching turned off:

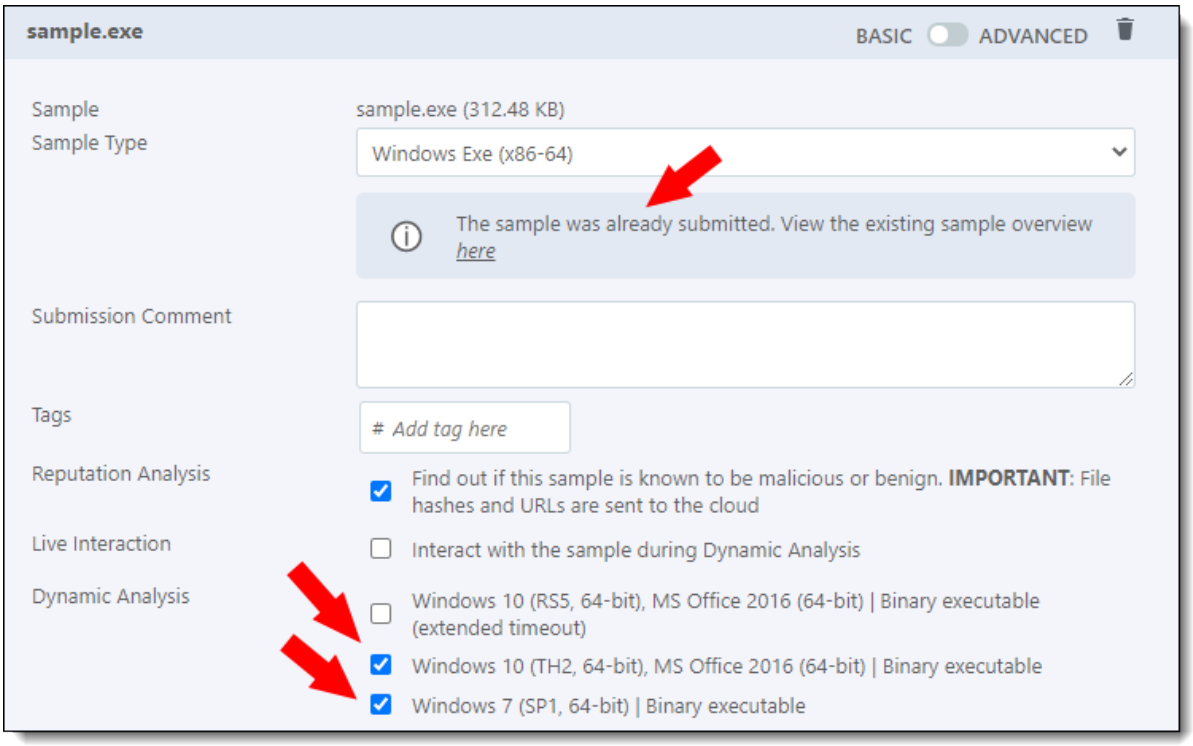
1. Log in to the Console.
2. Look at the current **Reports Usage** on the Dashboard. In this example it is **19 / 100**:



3. Select **Analysis Settings** from the Control Panel.
4. Select the **CONSOLE** button and verify that **Analysis Caching** is set to **Disabled**:

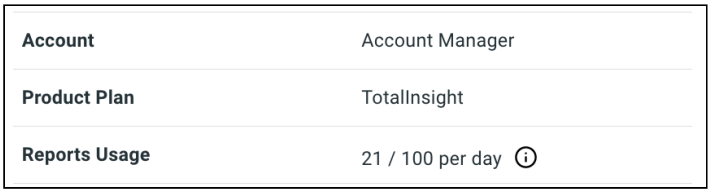


5. Return to the Upload Sample or URL page, and submit a **sample.exe** for analysis and the **Sample Submission** page displays. Note that there are two Dynamic Analysis environments defined for EXEs:

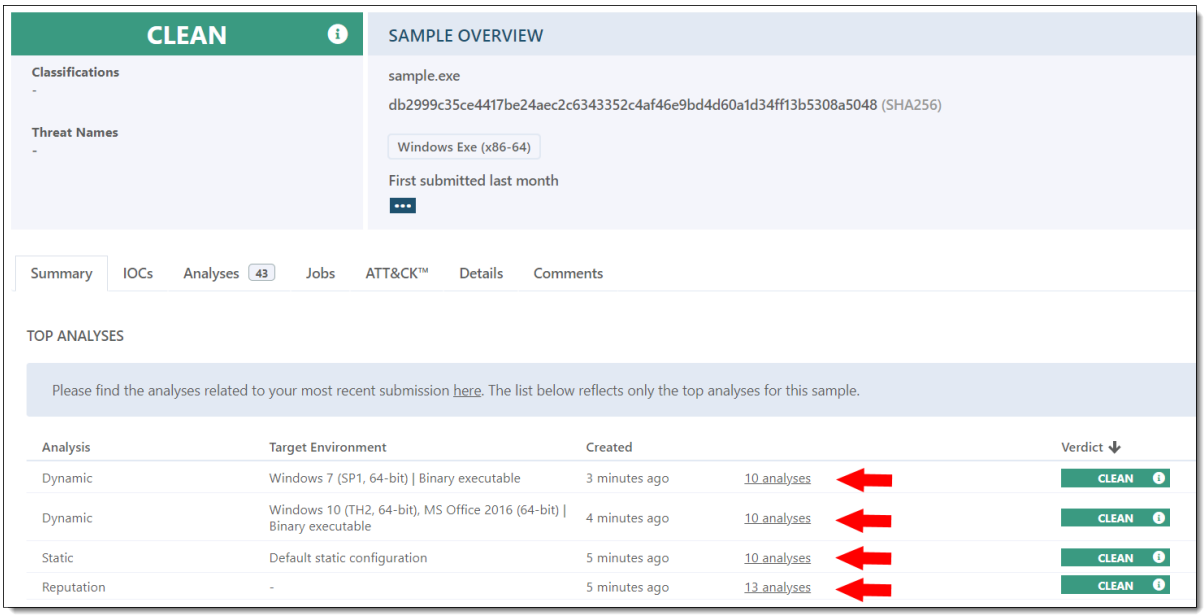


Also note that the information box indicates that **The sample was already submitted**.

6. Click **SUBMIT**.
7. Return to the Dashboard and you see that two quota items have been used:



8. Now look at the **SAMPLE OVERVIEW** to see a summary of the reports that have been created:



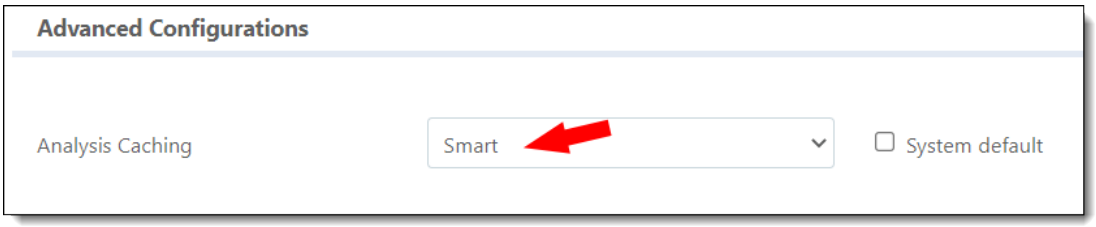
10 analyses have been created for the **Dynamic** Analysis in **Windows 7** and another **10 analyses** have been created for the **Dynamic** Analysis in **Windows 10**. Additionally, 10 Static Analyses have been created already for this file and 13 Reputation Analyses have been created. This is not very efficient to re-run analyses when the results are already cached. So now let's turn Smart Caching on and rerun the analysis of sample.exe.

12.2 Submitting with Smart Caching Enabled

Now let's turn Smart Caching on and submit the sample.exe file using the REST API:

1. Select **Analysis Settings** from the Control Panel menu.
2. Select **API KEYS**.
3. Locate your API key from the list that displays and click **Edit** in the **Actions** menu.
4. Scroll down to the **Advanced Configurations** section.

5. Select **Smart** from the **Analysis Caching** drop-down:

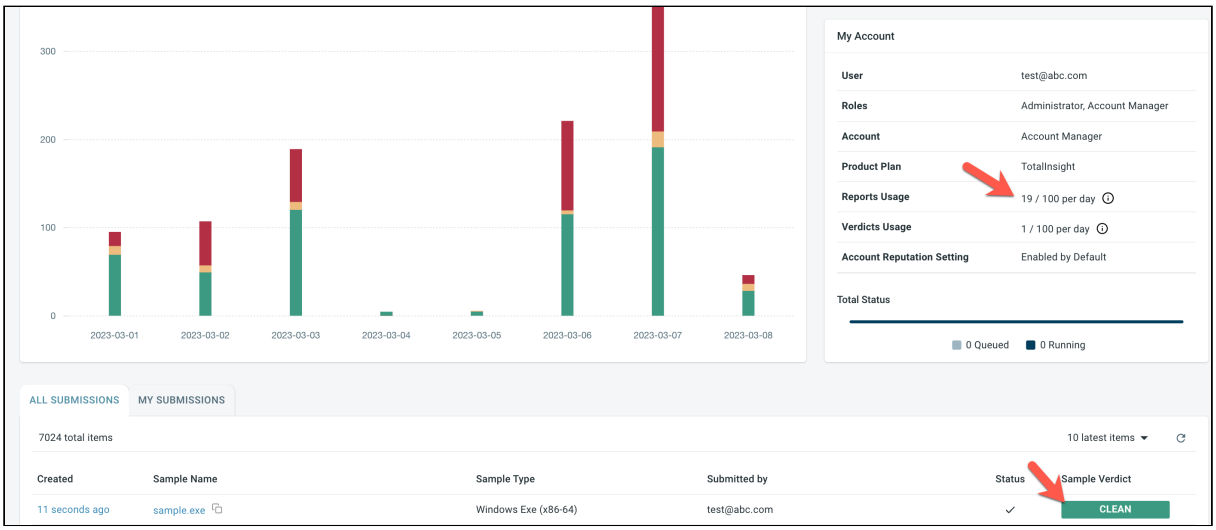


6. Click the **SAVE** button.

7. Submit the sample.exe with the curl that you used in earlier chapters and make sure to use the API key for which you just enabled smart Analysis Caching:

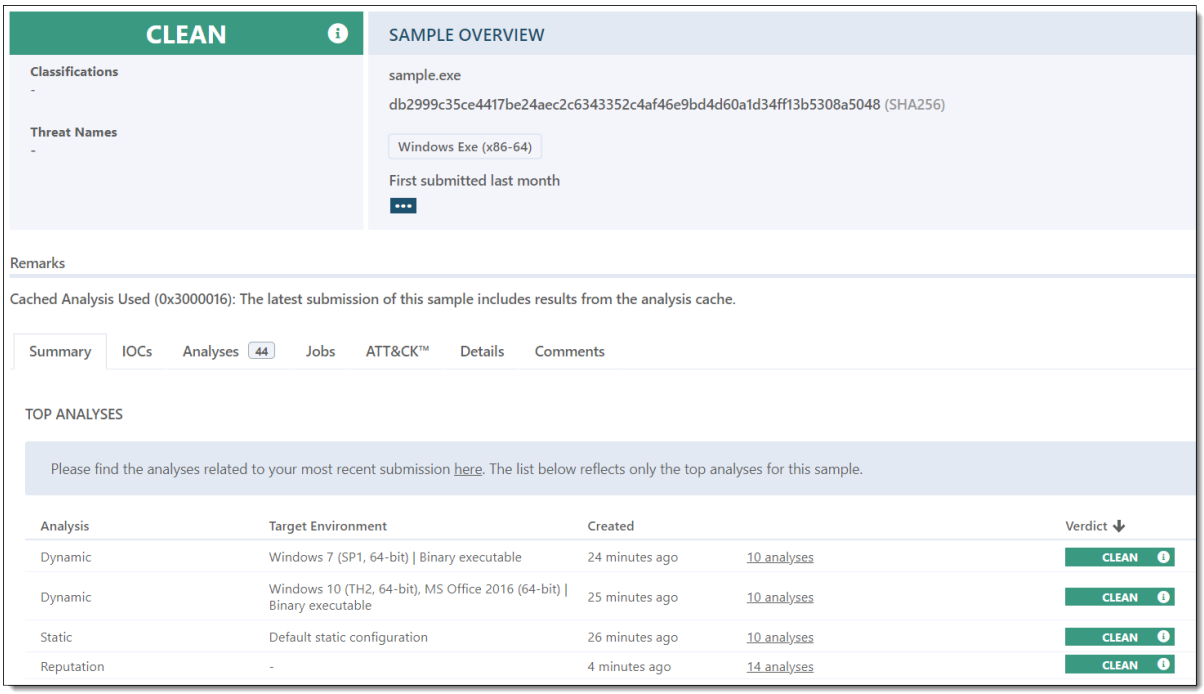
```
curl -X POST -H "Authorization: api_key juJs8ca1lCJaKaUbwcneKP5jRUjLJ0TF" -F
sample_file=@sample.exe "https://eu.cloud.vmray.com/rest/sample/submit"
```

8. Return to the Dashboard and you see that a Sample Verdict of **CLEAN** is returned immediately because the cached results have been used:



Notice that the **Report Quota Usage** has not gone up but has remained at **19 / 100**. This is the main reason to use Analysis Caching - there is no quota reduction at all because the analysis in the cache was used!

9. Look at the **SAMPLE OVERVIEW** to verify that no new Static or Dynamic Analyses were performed, and that the totals remain at 10 analyses each:



This demonstrates the utility of Analysis Caching. And this is a relatively simple example. Recall that some files, such as PDF files, can perform up to 20 different Dynamic Analyses. So if a single PDF file is submitted 10 times, your quota hit will be 200. You don't want that so make sure to always use Smart Analysis Caching.

Smart Analysis Caching uses a variety of factors to determine whether an analysis needs to be re-analyzed or not.

13 Tutorial: Understanding Webhooks

This chapter provides a step-by-step introduction on how to submit a sample with activated webhook notification. It also explains the webhook parameters. A webhook is an HTTP callback to a listener triggered by an event. Currently, the VMRay Platform supports emitting a webhook notification when all the analyses related to submission have been completed. The target of the webhook notification and additional details need to be specified per submission via the REST API.

The webhook concept presented in this chapter allows you to implement a lean and asynchronous sample submission workflow without any need for status polling.

On Premises administrators can enable webhook notifications. More information is provided in the *On Premises Installation Guide* > *Full Installation* > *Enabling Webhook Notifications* chapter.

13.1 Understanding the Webhook Parameters

To let the VMRay Platform send a webhook notification, you need to provide certain information about the target when submitting a sample. The **webhook_notification** parameter requires a dictionary along with the following entries:

- **url** - A string with the URL the webhook notification should be sent to. Must have a http or https scheme. It is strongly recommended to use https.
- **verify_tls** - An optional Boolean to control whether the server should verify the TLS certificate for a URL with https scheme. Defaults to true.
- **timeout** - An optional number of seconds that specifies the connection timeout to use for the webhook target. Defaults to a value set in the System Settings.
- **retries** - An optional integer that specifies the number of retries to attempt in case delivery of the webhook fails. Retries are attempted with an exponential backoff scheme. Defaults to a value set in the System Settings.
- **headers** - An optional dictionary that contains additional headers that should be set in the webhook notification

13.2 Submitting a Sample with Activated Webhook Notification

Use this call to submit the **sample.exe** for analysis and ensure that **webhook_notification** contains the URL of your web hook listener. In this example, an additional **X-AUTH** header is specified for additional security with a shared secret between submitter and webhook listener.

```
curl -X POST -H "Authorization: api_key SOME_SECRET_API_KEY" -F sample_file=@sample.exe -F
'webhook_notification={"url": "https://webhooks.test/queue", "headers": {"X-AUTH":
"SHARED_SECRET_FOR_ADDED_SECURITY"}}' "https://cloud.vmray.com/rest/sample/submit"
```

13.3 Webhook Specifics on Cloud

Webhooks sent from the Cloud Platform will be originating from the following IP addresses:

EU Cloud	US Cloud
3.122.137.24 18.185.19.121	44.223.214.141 52.205.240.109

Targets with IPv6-only addresses are not supported.

Targets must be using http or https on TCP port 80 or 443.

13.4 Understanding the Webhook Notification Format

When all related analyses to the submission of the **sample.exe** have been completed, a webhook notification request is sent to the specified target **url** using following format.

The webhook notification request will use the **POST** HTTP verb and include the specified headers. Following the example, the webhook listener can extract and evaluate the shared secret from the X-AUTH header. The body of the webhook notification request contains the following entries as JSON format.

- **submission_id** - ID of the submission.

- **sample_verdict** - Verdict for this submission.
- **sample_verdict_reason_code** - Verdict code for this submission.
- **sample_verdict_reason_description** - Verdict description for this submission.
- **sample_id** - ID of the sample.
- **sample_classifications** - List of classifications that have been detected.
- **sample_threat_names** - List of threat names that have been detected.
- **sample_console_url** - URL to the console page for this sample.
- **submission_status** - Whether the submission was successful or not.

Review this example of webhook notification request:

```
POST /queue HTTP/1.1
Host: webhooks.test:999
User-Agent: VMRayPlatform/4.4.0
X-AUTH: 839ba80438a914dc528435cf6f6700e
Content-Length: 317
Content-Type: application/json

{"submission_id": 2, "submission_status": "success", "sample_id": 1, "sample_classifications":
[], "sample_threat_names": [], "sample_console_url": "https://cloud.vmray.com/user/sample/view?
id=123", "sample_verdict": "clean", "sample_verdict_reason_code": null,
"sample_verdict_reason_description": null}
```

14 Installing and Using the Integration Kit

This document and the *API References* provide a wide variety of sample API calls that use the **curl** command line tool, which is the approach you should generally use for ad hoc use of the REST API. For more complicated tasks that require multiple steps to be implemented, such as submitting a sample and then waiting for all created jobs to finish, there is the Integration Kit. This chapter provides an overview of the Integration Kit and then describes how to install and use it. The next two chapters describe how to use the example python code provided with the Integration Kit.

14.1 An Overview of the Integration Kit

The Integration Kit is a Python 3.6+ library that helps you quickly build VMRay Platform integrations for typical use cases, and it accelerates your development efforts by providing ready-to-use python code, including code that helps you:

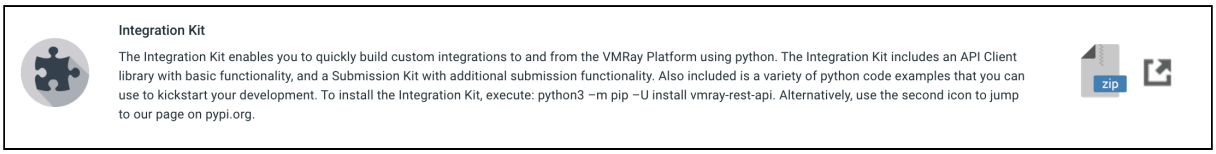
- Submit samples
- Retrieve analysis results, including the:
  - Severity Verdict
  - Verdict Reason
  - Classifications
  - Threat names
- Retrieve detailed analysis results, including:
  - VTIs
  - Artifacts
  - IOCs
- Retrieve child sample results generated by recursive analysis
- Download Summary.json v2 data, which has the analysis results
- Download all submissions as a stream

14.2 Installing the Integration Kit

To download the Integration Kit:

1. Open the Knowledge Center by clicking the **?** icon in the upper-right corner of the Console.
2. Scroll down to the **Platform Documentation for API Users** section.

3. Locate the **Integration Kit** area and you see that the install command is: **python3 –m pip install vmray-rest-api –U**



4. Run this command on your computer to install or update the Integration Kit.

> The **-U** indicates that an Update should be performed if you have an earlier version of the Integration Kit already installed.

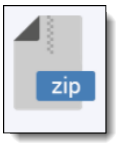
5. After the Integration Kit has been successfully installed, this message displays: **Successfully installed vmray-rest-api-5.1.1.** Of course, the version number (5.1.1) may be higher than the one you see here.

> The VMRay REST API files are stored very deep in your file hierarchy. For example: **c:\users\johnsmith\appdata\local\packages\pythonsoftwarefoundation.python.3.9_qbz5n2kfra8p0\localcache\local-packages\python39\site-packages**.

6. Alternatively, if you would like to see the Integration Kit repository on the **pypi.org** site, which has more information about the Integration Kit, click on the second icon:

The **tar.gz** file that you can download from the pypi site has an **examples** sub-folder with the 16 python code sample files.

7. As another alternative for obtaining the example code, you can click the ZIP icon to download just the 16 python code sample files:

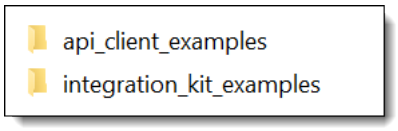


A ZIP file named **vmray-examples.zip** is downloaded.

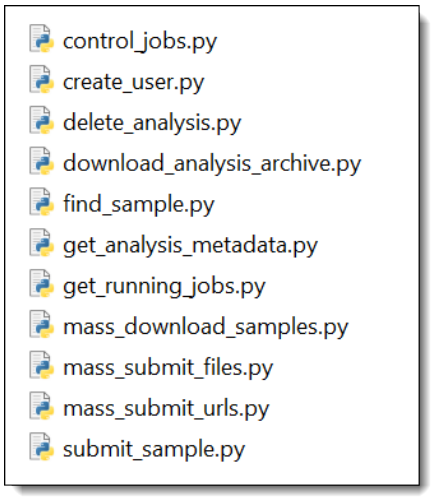
When developing your own integration using the Integration Kit, we recommend specifying the name and version in the optional parameter *connector_name.* This allows the Platform to identify calls from your connector.

14.3 Reviewing the Python Example Code

Regardless of which method you use to obtain the python code samples, you will find two sub-folders below the **examples** folder:

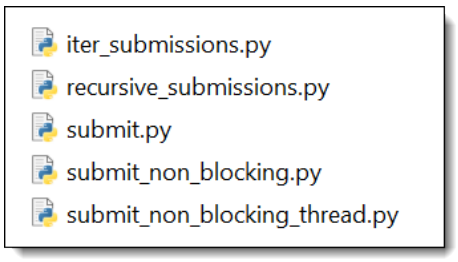


The first sub-folder, **api_client_examples** has these 11 python files:



These 11 code files are described in the Using the API Client Examples chapter.

The second sub-folder, **integration_kit_examples** has these 5 python files:



These 5 code files are described in the Using the Integration Kit Examples chapter.

14.4 Understanding the Two Required Calling Parameters

For both the API Client Examples and the Integration Kit examples, there are two parameters that you always need to pass:

- **-s** - **Your VMRay Platform REST API Host Endpoint.**
  For On Premises customers, this is your own proprietary URL where the Platform is installed. For Cloud customers, it is either eu.cloud.vmray.com or us.cloud.vmray.com. See the Creating API Calls chapter for more information.

- **-k** - **Your VMRay Platform REST API Key.**
  For both On Premises customers and Cloud customers, you set this key up yourself using the Console, as described in the Creating API Calls chapter.

Additional parameters for each specific example program can be seen in the code itself.

15 Using the API Client Examples

The **examples** folder, which you obtained in the previous chapter, has 11 sample python files in the **api_client_examples** folder. All 11 sample python files are briefly described in this chapter.

15.1 Submission Programs

The first category is Submissions and the following programs in this category are described:

- **submit_sample.py** - Submits a single sample to the VMRay Platform.
- **mass_submit_files.py** - Submits multiple files in a folder to the VMRay Platform.
- **mass_submit_urls.py** - Submits multiple URLs in a folder to the VMRay Platform.

15.2 Download Programs

The second category is for code that enables you to download data from the VMRay Platform:

- **download_analysis_archive.py** - Downloads an Analysis Archive, which is a large ZIP file with all artifacts, IOCs, reports and other information generated during analysis.
- **mass_download_samples.py** - Downloads multiple samples and stores them as ZIP Files with a password of: infected.

Samples are typically harmful malware so always exercise caution when downloading them and when opening ZIP files that contain them.

15.3 Job Programs

The third category relates to jobs:

- **get_running_jobs.py** - Obtains a list of all currently running jobs.
- **control_jobs.py** - Enables you to set a variety of job parameters, including the minimum priority, waiting for the job queue to be empty, and waiting for the inwork queue to be empty.

15.4 Maintenance Programs

And the fourth category has a variety of programs for different maintenance purposes:

- **find_sample.py** - Finds a single sample based on its SHA256 value.
- **create_user.py** - Creates a new user by passing the following parameters: email, password, company, first name, last name, and optionally, a SamplesetID.
- **delete_analysis.py** - Deletes an analysis based on its Analysis ID.
- **get_analysis_metadata.py** - Obtains all metadata related to a Dynamic Analysis that was performed on the VMRay Platform.

16 Using the Integration Kit Examples

The **examples** folder, which you obtained in the Installing and Using the Integration Kit chapter, has 5 sample python files in the **integration_kit_examples** folder. All 5 sample python files are briefly described in this chapter.

16.1 Understanding the Five Example Code Files

These 5 sample python files are a great starting point for interacting with the VMRay Platform:

1. **submit.py** - Submits a sample file and gathers results, such as the Sample Verdict (in blocking mode).
2. **submit_non_blocking.py** - Submits a sample file and gathers results (in non-blocking mode).
3. **submit_non_blocking_thread.py** - Submits multiple files and gathers all results in different threads.
4. **recursive_submission.py** - Gets all results for a recursive submission.
5. **iter_submissions.py** - Iterates over completed submissions to the API.

16.2 Looking at an Example Call

In addition to the **-s** parameter (which has your host endpoint) and the **-k** parameter (which has your API key), each call has additional parameters you can pass, such as the **-f** parameter for **submit.py** which is where you specify the file name to submit. So, a typical call for **submit.py** is set up like this:

```
python integration_kit_examples/submit.py -s https://cloud.vmray.com -k YOUR_API_KEY -f /path/
to/sample_file
```

For example, this call submits a file named **sample.exe**, which is stored in the **C:\Users\johnsmith** directory using an API key on the European (**eu**) Cloud HTTP endpoint:

```
python integration_kit_examples/submit.py -s https://eu.cloud.vmray.com -k
xj2MG4NCgs9EkQVsThTSjAZvfuXqtl8 -f C:\Users\johnsmith\sample.exe
```

16.3 Getting Help

You can view all parameters for any of the Python samples by invoking the script with **–h or --help**. For example:

```
python integration_kit_examples/submit.py -h
```

Returns:

```
usage: submit.py [-h] -s SERVER -k API_KEY -f SAMPLE_FILE

Example to submit a sample to VMRay Platform.

optional arguments:
  -h, --help            show this help message and exit
  -s SERVER, --server SERVER
                        VMRay Platform server address.
  -k API_KEY, --api_key API_KEY
                        VMRay Platform API key.
  -f SAMPLE_FILE, --sample_file SAMPLE_FILE
                        A sample file to submit to VMRay Platform.
```