API Guide for Content Analysis and Malware Analysis

CA Version 2.3

Legal Notice

Symantec Corporation
350 Ellis Street
Mountain View, CA 94043

www.symantec.com

2/7/2018

Table of Contents

Content Analysis APIs

Content Analysis offers three Application Programming Interfaces (APIs):

- REST Application Programming Interface for submitting files for scanning
- REST Application Programming Interface for performing malware analysis functions
- Publish-Subscribe Application Programming Interface (Pub-Sub API) to provide notification about tasks and scanning verdicts

This manual is intended for malware analysts, researchers, and security practitioners who are using Content Analysis for content scanning and malware analysis. This manual assumes that the reader is well versed in network terminology and operations, and is familiar with malware in general and malware analysis in particular. An understanding of Windows system events and network intrusion techniques is helpful as well.

Generate an API Key

Authentication to the API is provided using unique authentication tokens (API keys) matched to specific roles and access privileges.

Example API key: `3970ae9b89b6402da1b706435d56006c`

Administrators can create and manage API keys in the Content Analysis CLI interface. To generate an API key:

1. Connect to the serial console or SSH to the Content Analysis appliance as a user with administrative privileges.
2. Enter the boldfaced commands below:

   ```
   > enable

   Password: <enter the password>

   #ma-actions api-key create administrator

   Note that keys are not stored on the system in plain text and cannot be retrieved
   later.

   Created new API Key: <This is the API key> (Key ID 2)
   ```

3. Copy the generated API key and save it in a text file, as it cannot be viewed later.

Other CLI commands are available to view and delete API keys:

- **`ma-actions api-key list`** Shows the ID for each API key and its associated privileges. Note that it does NOT show the value of the key.
- **`ma-actions api-key delete <id>`** Deletes the API key with the specified ID.

Lost API Keys

API keys are not stored in clear text in the system database. For security reasons, only a one-way hash value is stored. If a key is lost, it is impossible to retrieve it. In this situation, a user with an Administrator or a Super Analyst role should delete the lost key (if it is identifiable via the UID and role), or delete and recreate all keys belonging to that user.

When a new key is created, the key is communicated to the administrative user in the output of the **`ma-actions api-key create`** CLI command. The administrator must then provide the user with the key via an external mechanism and it is the user's responsibility to securely store their key external to Content Analysis.

API Client Environment

The examples in this document use *cURL*, an open-source library and command-line tool for transferring data using various protocols with URL syntax. It supports DICT, FILE, FTP, FTPS, Gopher, HTTP, HTTPS, IMAP, IMAPS, LDAP, LDAPS, POP3, POP3S, RTMP, RTSP, SCP, SFTP, SMTP, SMTPS, Telnet and TFTP. cURL also supports SSL certificates, HTTP POST, HTTP PUT, FTP uploading, HTTP form based upload, proxies, cookies, and user/password authentication.

The examples use cURL because it's easier to demonstrate API usage through the cURL command-line tool rather than to include source code. However, you can access the Content Analysis and Malware Analysis APIs with your preferred coding tool.

Pass API Keys to REST API

Once you have generated an API key, you can use it to authenticate REST API calls. To append an 'X-API-TOKEN' header for HTTP requests, the cURL command would be:

```
curl -H "X-API-TOKEN:<api-key>" https://mag2host/rapi/tasks
```

For example, if the API key is a2f26f1c84084eb4b1c4a694aa8ae9e6:

```
curl -H "X-API-TOKEN:a2f26f1c84084eb4b1c4a694aa8ae9e6" https://mag2host/rapi/tasks
```

REST API for Malware Analysis Operations

The full-featured Malware Analysis Application Programming Interface (API) can perform the operations in the Malware Analysis tab as well as additional functionality.

The API supports basic and advanced integration and automation goals including: multi-step workflow processes; integration into anti-malware solution suites; integration into extended security infrastructures; and context, correlation, and enrichment processes. Content Analysis supports generic data exchange with external systems via a well-defined event schema that is accessible through the API via Python, cURL, Ruby, and many other common programming languages.

Common API uses include:

- Bulk file submission for analysis
- Good-bad file determination for white/black lists
- Submit custom tasks to multiple analysis environments (SandBox, IntelliVM profile, MobileVM profile)
- Assign priorities to sample tasks (High, Medium, Low)
- Automatically submit dropped files for analysis
- Conditional processing based on risk scores or behavioral indicators
- Post-processing of analysis artifacts using third-party tools
- Inclusion of other tools as part of the core analysis process (such as VirusTotal counts, YARA signatures)
- Back-end for other processes (for example, checking attachments before uploading to Salesforce.com)
- Integration into coordinated product suites with other tools and appliances
- Enhance analysis with context, correlation, and enrichment processes

Common Malware Analysis API Functions

The Malware Analysis API enables a rich array of dynamic functionality that is limited only by programming ability and imagination.

It is impractical to detail all possible API usage implementations, so we provide a diverse, core set of examples to enable productive usage of API functionality and jump-start learning of higher-level tasks.

For a complete list of Malware Analysis API functions, open the Content Analysis context-sensitive help and search for API.

Shut Down or Restart System

Shut down or restart the Content Analysis appliance.

Syntax

```
$ curl -X POST https://<CA-host>/rapi/system/shutdown
```

```
$ curl -X POST https://<CA-host>/rapi/system/reboot
```

HTTP Return Value

JSON

HTTP URL

`/system/shutdown`

`/system/reboot`

Submit Sample File

Submit a new basic sample. Parameters must be encoded using multipart/form-data and not application/x-www-form-urlencoded.

Syntax

```
$ curl -X POST --form upload=@<filename> --form "owner=admin" --form
   "extension=jpg" https://<CA-host>/rapi/samples/basic
```

HTTP Return Value

`JSON`

HTTP Parameters

source	Source of sample. Default = www
resource_id	Create sample using existing sample_resource
file	Multipart form-data file attachment
owner	Owner of sample (required)
label	Label of sample
target_name	Override file name
description	Free-form text description
extension	Override file extension
exec_arguments	Override default execution arguments

HTTP URL

`/samples/basic`

Submit Sample URL

Submit a sample URL.

Syntax

```
$ curl -X POST -d "url=http://<URL>/" https://<CA-host>/rapi/samples/url
```

HTTP Return Value

`JSON`

HTTP Parameters

source	Source of sample. Default = www
exec_arguments	Override default execution arguments
owner	Owner of sample (required)
label	Label of sample
description	Free-form text description
url	URL of sample

HTTP URL

`/samples/url`

Create Task

Create a new task. A *task* is an execution of a sample file or URL in a defined environment (operating system profile + testing plugin script).

Syntax

$ curl -X POST -d "sample_id=*<sample_id>*&env=drd" https://*<CA-host>*/rapi/tasks

$ curl -X POST -d "sample_id=*<sample_id>*&env=sbx&tp_IVM.TIMEOUT=30" https://*<CA-host>*/rapi/tasks

$ curl -X POST -d "sample_id=*<sample_id>*&env=ivm&exec_args=c:\windows\wscript.exe {sample}" https://*<CA-host>*/rapi/tasks

HTTP Return Value

`JSON`

HTTP Parameters

sample_id	Sample ID (required)
storage_classes	How the event data will be stored. 1 Local database 2 Local fileshare 4 Local fileshare as GZIP 8 Amazon S3 cloud storage 16 Amazon S3 cloud storage as GZIP
env={sbx|ivm|drd}	Task environment (required)
primary_resource_id	Override default execution resource by ID
ivm_profile	Virtual machine profile short name, if `env=ivm` is specified
exec_args	Override default execution arguments
primary_resource_name	Override default execution resource by name

priority={high|medium|low}	Task priority
tp_<task_property>	Set multiple task properties
vmp_id	Virtual machine profile ID, if `env=ivm` is specified

HTTP URL

`/tasks`

Get Task Statistics

Get statistics for a completed task.

Syntax

$ curl -X GET https://*<CA-host>*/rapi/tasks/*<task_id>*/stats

HTTP Return Value

`JSON`

HTTP URL

`/tasks/(?P<task_id>[09]+)/stats`

Get a Sample's Tasks

Get tasks for a sample.

Syntax

$ curl X GET https://*<CA-host>*/rapi/samples/*<sample_id>*/tasks

HTTP Return Value

`JSON`

HTTP URL

`/samples/(?P<sample_id>[09]+)/tasks`

Get Pattern Group

Get all pattern groups or a specific pattern.

Syntax

$ curl -X GET https://*<CA-host>*//rapi/pattern_groups

$ curl -X GET https://*<CA-host>*//rapi/pattern_groups/*<group_ID>*

HTTP Return Value

JSON

HTTP Parameters

include_global	Include/exclude global patterns. Default=1 (include).
owner	Return only patterns belonging to the specified owner. If owner contains a value, include_global=0 must also be specified.

HTTP URL

/pattern_groups

/pattern_groups/(?P<*pattern_group_id*>[09]+)

/pattern_groups/(?P<*pattern_group_uuid*>[af09\]{36})

Retrieve Sample Tasks

Get tasks for a sample.

Syntax

$ curl -X GET https://<*CA-host*>/rapi/samples/<*task_id*>/tasks

HTTP Return Value

JSON

HTTP URL

/samples/(?P<*sample_id*>[09]+)/tasks

Get Risk Score

Get the risk score of a completed task.

Syntax

$ curl -X GET https://<*CA-host*>/rapi/tasks/<*task_id*>/risk_score

HTTP Return Value

JSON

HTTP Parameters

owner	override sample owner for risk calculation

HTTP URL

`/tasks/(?P<task_id>[09]+)/risk_score`

Retrieve Matched Patterns

Get pattern group results for a completed task.

Syntax

`$ curl -X GET https://<CA-host>/rapi/tasks/<task_id>/patterns`

HTTP Return Value

`JSON`

HTTP URL

`/tasks/(?P<task_id>[09]+)/patterns`

Retrieve Task Events

Get event data for a completed task.

Syntax

`$ curl -X GET https://<CA-host>//rapi/tasks/<task_id>/events`

`$ curl -X GET https://<CA-host>//rapi/tasks/<task_id>/events?mode=gpb&include_norm_stats=1`

HTTP Return Value

`[JSON|GPB]`

HTTP Parameters

`include_norm_stats`	Include/exclude normalization statistics. Default=0 (exclude).
`event_filter`	Return only certain event types (not implemented)
`mode={json|gpb}`	Return events as JSON or Google Protocol Buffer. Default=json.

HTTP URL

`/tasks/(?P<task_id>[09]+)/events`

Retrieve Task Resources

Get resources for a completed task.

Syntax

```
$ curl -X GET https://<CA-host>/rapi/tasks/<task_id>/resources
```

HTTP Return Value

JSON

HTTP URL

/tasks/(?P<task_id>[09]+)/resources

Retrieve Sample Binary File

Get sample resource binary.

Syntax

```
$ curl -X GET https://<CA-host>/rapi/samples/resources/<resource_id>/bin
```

HTTP Return Value

binary

HTTP URL

/samples/resources/(?P<resource_id>[09]+)/bin

Search for a Sample

Search for a basic sample.

Syntax

```
$ curl -X GET https://<CA-host>/rapi/samples/basic/<sample_id>
$ curl -X GET https://<CA-host>/rapi/samples/basic?owner=admin
$ curl -X GET https://<CA-host>/rapi/samples/basic?owner=admin&limit=2&offset=10
```

HTTP Return Value

JSON

HTTP Parameters

sha256	Search by SHA256
exact	If exact=0 then allow partial string matches. Default=0.
md5	Search by MD5
owner	Search by owner

lable	Search by label
offset	Combine with `limit` to page through results. Default=0.
limit	Limit number of results. Default=100.
resource_magic	Search by sample resource magic string
resource_name	Search by sample resource name

HTTP URL

`/samples/basic`

`/samples/basic/(?P<sample_id>[09]+)`

Delete Task

Deletes a task. All events and task resources are also deleted.

Syntax

`$ curl -X DELETE https://<CA-host>/rapi/tasks/<task_id>`

HTTP Return Value

`JSON`

HTTP URL

`/tasks/(?P<task_id>[09]+)`

Delete Sample

Delete a sample. This deletes the sample record, all attached task records, task events, task resources, and sample resources. Sample resources records will not be deleted if another sample also shares the sample resource. Sample resource binaries will be deleted and the resource flag set to False.

Syntax

`$ curl -X DELETE https://<CA-host>/rapi/samples/<sample_id>`

HTTP Return Value

`JSON`

HTTP URL

`/samples/(?P<sample_id>[09]+)`

Fetch VirusTotal Data

VirusTotal is a free virus, malware, and URL online scanning service.

Syntax

```
$ curl -X GET https://<CA-host>/rapi/3rdparty/vt/<file_md5>
```

where *<file_md5>* is the MD5 hash of the file. For example:

```
$ curl -X GET https://<CA-host>/rapi/3rdparty/vt/6600aaf7babed63bca0fa860ff0f69ff
```

HTTP Return Value

JSON

HTTP URL

```
/3rdparty/vt/(?P<md5>[afAF09]{32})
```

Get Status of Queues

Get status of files in the queue waiting for the on-box sandbox to analyze.

Syntax

```
$ curl -X GET https://<CA-host>/rapi/system/queues
```

HTTP Return Value

JSON

HTTP URL

```
/system/queues
```

Get Health State

Get health state (green/yellow/red) of the Content Analysis appliance.

Syntax

```
$ curl -X GET https://<CA-host>/rapi/system/health
```

HTTP Return Value

JSON

```
health_states = {0: "green", # Full operational 1: "yellow", # Operational, needs
   maintenance 2: "red" # Not operational, needs immediately inspection}
```

HTTP URL

```
/system/health
```

Get Database Counts

Get database counts, including sample count, task count, event count, task count grouped by environment, task count grouped by status.

Syntax

```
$ curl -X GET https://<CA-host>/rapi/system/stats/counts
```

HTTP Return Value

```
JSON
```

HTTP URL

```
/system/stats/counts
```

List Current Sessions

List all current user sessions. As an alternative to using an API key to authenticate, users can authenticate with their Content Analysis user name and password, which will provide them with a temporary API key (also called a token).

Syntax

```
$ curl -X GET https://<CA-host>/rapi/auth/sessions
```

HTTP Return Value

```
JSON
```

HTTP URL

```
/auth/sessions
```

Download Windows Base Image

Download a Windows base image from the specified Content Analysis appliance and place it on a local system.

Syntax

```
$ curl -OJ https://<CA-host>/rapi/system/vm/bases/<vmb_id>/bin
```

```
$ curl -k -OJ -H "X-API-TOKEN:<API-key>" https://<CA-
  host>:8082/rapi/system/vm/bases/<vmb_id>/bin
```

HTTP Return Value

```
Binary
```

Example

```
$ curl -k -OJ -H "X-API-TOKEN:7a49af86645e4e3a9f24608636135f64"
https://10.10.10.10:8082/rapi/system/vm/bases/1/bin
```

HTTP URL

```
/system/vm/bases/<vmb_id>/bin
```

```
/system/vm/bases/(?P<vmb_id>[0-9]+)/bin
```

Pull Windows Base Image from URL

Import a Windows base image onto the specified Content Analysis appliance by pulling the file from the specified URL.

Syntax

```
$ curl -d 'url=http://web_server/base_image.bundle' https://<CA-
host>/rapi/system/vm/bases/pull?product_key=<key>
```

HTTP Parameters

api_key	API key when downloading from Content Analysis (optional)
product_key	Microsoft Windows product key for the IVM
validate_cert	Validate HTTPS certificate. Default=1.
retry_failed	Retry base image download if the name is the same and the state is error. Default=0.
use_proxy	Use configured proxy server. Default=0.
url	Location of VM bundle (required)
build_profile	Build standard profile after downloading image. Default=1.

HTTP Return Value

```
JSON
```

Example

```
$ curl -k -H "X-API-TOKEN:7a49af86645e4e3a9f24608636135f64" -d 'url=http://web_
server/base_image.bundle'
https://10.10.10.10:8082/rapi/system/vm/bases/pull?product_key=xxxxx-xxxxx-xxxxx-
xxxxx-xxxxx
```

HTTP URL

```
/system/vm/bases/pull
```

Upload Windows Base Image

Upload a Windows base image to the specified Content Analysis appliance.

Syntax

```
$ curl -F upload=@base_image.bundle https://<CA-
host>/rapi/system/vm/bases/post?product_key=<key>
```

HTTP Parameters

product_key	Microsoft Windows product key for the IVM
file	Multi-part form-data file attachment
build_profile	Build a standard profile after upload. Default=1.

HTTP Return Value

JSON

Example

```
$ curl -k -H "X-API-TOKEN:7a49af86645e4e3a9f24608636135f64" -F upload=@base_
image.bundle https://10.10.10.10:8082/rapi/system/vm/bases/post?product_key=xxxxx-
xxxxx-xxxxx-xxxxx-xxxxx
```

HTTP URL

```
/system/vm/bases/post
```

Pull Windows ISO from URL

Install a Windows ISO image on to the specified Content Analysis appliance by pulling it from the specified URL.

Syntax

```
$ curl -d 'url=http://web_server/windows.iso' https://<CA-
host>/rapi/system/vm/bases/pull_iso?product_key=<key>&iso_type=<type>
```

HTTP Parameters

product key	Microsoft Windows product key for the IVM
display_name	Display name of the new base image (required)
validate_cert	Validate HTTPS certificate. Default=1.
use_proxy	Use configured proxy server. Default=0.
file	Multi-part form-data file attachment

iso_type={win7x64|win10x64}	Type of ISO (required)
build_profile	Build standard profile after downloading image. Default=1.

HTTP Return Value

JSON

Example

```
$ curl -k -H "X-API-TOKEN:7a49af86645e4e3a9f24608636135f64" -d 'url=http://web_
server/windows.iso' https://10.10.10.10:8082/rapi/system/vm/bases/pull_iso?product_
key=xxxxx-xxxxx-xxxxx-xxxxx-xxxxx&iso_type=win10x64
```

HTTP URL

```
/system/vm/bases/pull_iso
```

Upload Windows ISO File

Upload and install a Windows ISO image to the specified Content Analysis appliance.

Syntax

```
$ curl -F file=@windows.iso https://<CA-host>/rapi/system/vm/bases/post_
iso?product_key=<key>&iso_type=<type>
```

HTTP Parameters

product_key	Microsoft Windows product key for the IVM
display_name	Display name of the new base image (required)
file	Multi-part form-data file attachment
iso_type={win7x64|win10x64}	Type of ISO (required)
build_profile	Build a standard profile after upload. Default=1.

HTTP Return Value

JSON

Example

```
$ curl -k -H "X-API-TOKEN: 3b6d132a0d6b44409693d55d77137a8c"
"https://10.10.10.10:8082/rapi/system/vm/bases/post_iso?build_profile=1&product_
key=xxxxx-xxxxx-xxxxx-xxxxx-xxxxx&name=win7x64_post&display_name=win7x64_post&iso_
type=win7x64" \

-F file=@WIN7-x17.iso"
```

HTTP URL

`/system/vm/bases/post_iso`

Export IVM Profile

Export an IVM profile or modify an already exported profile's metadata.

Syntax

```
$ curl -X POST -d \"vmp_id=1\" https://<CA-host>/rapi/system/vm/profiles/export
```

```
$ curl -X POST -d \"description=new description\" https://<CA-
   host>/rapi/system/vm/profiles/export/05E9QMMKF8ANPGZNG7DJY9D4DR
```

HTTP Return Value

`JSON`

HTTP Parameters

queue_if_busy	Queue the operation if controller is busy. Default=0 (off)
short_name	Abbreviated name of the exported profile
vmp_id	Virtual machine profile ID to export (only if `vme_id` is not specified)
compression={none|gzip|lzma}	Optional compression to use. Default=none.
name	Name of the exported profile
description	Description of the exported profile
ttl	Number of hours the exported profile should be kept on disk. Default=24. Enter 0 to keep forever.

HTTP URL

`/system/vm/profiles/export`

`/system/vm/profiles/export/(?P<vme_id>[0-9A-HJKMNPQRSTVWXYZ]{26})`

Get Exported IVM Profile Metadata

Get exported IVM profile metadata.

Syntax

```
$ $ curl -X GET https://<CA-
   host>/rapi/system/vm/profiles/export/05E9QMMKF8ANPGZNG7DJY9D4DR
```

HTTP Return Value

`JSON`

HTTP Parameters

vme_id	Exported IVM profile ID (optional)

HTTP URL

```
/system/vm/profiles/export
```

```
/system/vm/profiles/export/(?P<vme_id>[0-9A-HJKMNPQRSTVWXYZ]{26})
```

Download Exported IVM Profile

Download an exported IVM profile to the specified CA appliance.

Syntax

```
$ curl -X GET https://<CA-
   host>/rapi/system/vm/profiles/export/05E9QMMKF8ANPGZNG7DJY9D4DR/bin
```

HTTP Return Value

```
Binary
```

HTTP URL

```
/system/vm/profiles/export/(?P<vme_id>[0-9A-HJKMNPQRSTVWXYZ]{26})/bin
```

Import IVM Profile

Import an exported IVM profile.

Syntax

```
$ curl -X POST -F upload=@profile.qcow2.bundle https://<CA-
   host>/rapi/system/vm/profiles/import
```

HTTP Return Value

```
JSON
```

HTTP Parameters

short_name	Abbreviated name to use for the new profile (default from the bundle)
file	Multi-part form-data file attachment
name	Name to use for the new profile (default from the bundle)
set_default	Set the profile as the default profile
build_profile	Build the profile after import. Default=1 (yes)
overwrite	Overwrite the profile if it already exists. Default=0 (off)
description	Description of the exported profile

HTTP URL

```
/system/vm/profiles/import
```

Pub-Sub API for Notifications

The Publish-Subscribe Application Programming Interface (Pub-Sub API) allows for the immediate and secure notification of tasks in progress, tasks that have been completed, and scanning verdicts. This functionality allows for the automation of additional post-processing outside of Content Analysis, the initiation of downstream processes in other systems, and for the assignment of targeted actions to be taken by analysts or security practitioners.

WebSocket is a web technology providing full-duplex communications channels over a single TCP connection. WebSocket differs from TCP in that it enables a stream of messages instead of a stream of bytes. The communications are done over TCP port numbers 8081 and 8082, which is of benefit for those environments that block non-standard Internet connections using a firewall.

The WebSocket implementation provides for secure communications between clients and servers over the web. It features an HTTP-compatible handshake so that HTTP servers can share their default HTTP and HTTPS ports with a WebSocket gateway or server.

The WebSocket protocol uses ws:// and wss:// prefixes to indicate standard WebSocket and WebSocket Secure connections, respectively. Symantec strongly advises its customers to utilize WebSocket Secure wss://.

Pass API Keys to the Pub-Sub API

The Pub-Sub API accepts API keys for user security authentication. (See "Generate an API Key" on page 4.) This simple example in Python demonstrates how to connect and authenticate with the Pub-Sub API:

```
from websocket import create_connection
addr ="wss://CA_HOST:8082/rapi/ws/task_state"
ws = create_connection(addr, header={'X-API-TOKEN': <api-key>})
while processing_tasks:
  print ws.recv()
```

WebSockets and Commands

The `task_complete` and `task_state` WebSockets accept JSON-formatted commands that identify which tasks the client wishes to receive notifications for.

Supported commands and formats include:

`{"command":"add_task", "args":"*"}`	Receive notifications for all tasks
`{"command":"add_task", "args":<task_id>}`	Receive notifications for the specified task
`{"command":"add_task", "args":[<task_id>, <task_id>]}`	Receive notifications for all tasks in the list
`{"command":"clear_tasks", "args":None}`	Clear all tasks from the list

task_complete

The `task_complete` WebSocket retrieves notifications when tasks are complete. This WebSocket provides two

URL endpoints:

ws[s]://<CA-host>/rapi/ws/task_complete/*

Connections to this endpoint will receive task_ complete notifications for all tasks processed by the system. However, if authentication is enabled, only users with privileges to view all tasks can connect to this endpoint.

ws[s]://<CA-host>/rapi/ws/task_complete

Connections to this endpoint require additional commands, in the format shown above, to indicate which tasks the user wishes to receive notifications for. If authentication is enabled, users can receive only notifications for tasks that they have permission to view.

This WebSocket will not provide notifications for failed tasks. The task_state WebSocket should be used to receive notifications of any tasks that enter an error state.

task_state

The `task_state` WebSocket is used to receive notifications when a task changes state. This WebSocket provides two different URL endpoints:

ws[s]://<CA-host>/rapi/ws/task_state/*

Connections to this endpoint will receive `task_ state` change notifications for all tasks processed by the system. However, if authentication is enabled, only users with privileges to view all tasks can connect to this endpoint.

ws[s]://<CA-host>/rapi/ws/task_state

Connections to this endpoint require additional commands, in the format detailed above, to indicate which tasks the client wishes to receive notifications for. If authentication is enabled, the client can only receive notifications for tasks that it has privileges to view.

Example

The notifications are JSON-formatted text and include only the task ID and state value, as shown in the following example:

```
'{"new_state": 2, "task_id": 443}'
```

A list of all valid `task_state` values can be found in "Task States" below.

Task States

The table below lists valid task states for on-box sandboxing.

Task State	Description	API task_state ID
CORE_UNINITIALIZED	Task is uninitialized	Task_State = 0
CORE_INITIALIZED	Task is initialized	Task_State = 1
CORE_INTASKQUEUE	Task is queued	Task_State = 2
CORE_POSTPROCESSING	Task event processing	Task_State = 3

Task State	Description	API task_state ID
CORE_ININSERTQUEUE	Task events are awaiting insert	Task_State = 4
CORE_ERROR	Generic error	Task_State = 5
CORE_COMPLETE	Task complete	Task_State = 6
CORE_PROCESSING_INSERT	Inserting events in database	Task_State = 7
CORE_INSERT_ERROR	Error inserting events in database	Task_State = 8
SBX_INPROCESS	Processing task in SandBox or MobileVM	Task_State = 100
SBX_COMPLETE	SandBox or MobileVM task is complete	Task_State = 101
SBX_ERROR	Error while processing in SandBox or MobileVM	Task_State = 102
IVM_INPROCESS	Processing task in IntelliVM	Task_State = 200
IVM_COMPLETE	IntelliVM task is complete	Task_State = 201
IVM_ERROR	Error while processing in IntelliVM	Task_State = 202
APPLE_INPROCESS	Processing task in Apple Analyzer	Task_State = 300
APPLE_COMPLETE	Apple Analyzer task is complete	Task_State = 301
APPLE_ERROR	Error while processing in Apple Analyzer	Task_State = 302

syslog

The syslog WebSocket receives notifications for all syslog entries that are emitted by the system's malware analysis components.

ws[s]://*<CA-host>*/rapi/ws/syslog[?loglevel=*<value>*]

The optional `loglevel` parameter indicates the level of messages that will be received, as defined below:

```
_log_levels = {
      'ERR': 3,
      'WARNING': 4,
      'INFO': 6,
      'DEBUG': 7
}
```

If `loglevel` is omitted, this value defaults to the config value set for bootstrap.loglevel.

Example

```
{"source": "stats", "message": "Starting stats update", "level": 7, "ts": "2014-12-
  10T02:37:40.011019", "server": "mag2"}
```

health

This WebSocket receives notifications about the Content Analysis system health status.

ws[s]://*<CA-host>*/rapi/ws/health

A message is published for each health check, indicating the reason for the check and the current health state (0=green, 1=yellow, 2=red).

Examples

```
{"last_state": 0, "msg_type": "state_unaltered", "state": 0, "source": "df_health",
   "reason": "df_health_daemon_not_running", "time": 1418178554}
```

```
{"last_state": 2, "msg_type": "state_unaltered", "state": 2, "source": "license",
   "reason": "license_validity", "time": 1418178886}
```

Sample Use Cases for Pub-Sub API

The Pub-Sub API facilitates many productive use cases for both end users and integrators, including post-processing and conditional processing of task results. Some common examples are described below. Notification via the Pub-Sub API makes it possible to automate processing as soon as the desired results become available.

Resubmit Dropped Files

Malware often proceeds along a multi-stage infection cycle, where an initial file drops additional files during its run. These additional files may be of interest to the analyst, who can gather up these resources and resubmit them for automated analysis in Content Analysis.

In the API, use the *<task_id>* to request `task_resources`. Request each task resource, iterate through the list to identify resources of interest, download to a local file system, and then submit the desired files normally. Task resources may include screen shots, PCAPs, dropped files, and files the plugin itself creates.

Syntax

To fetch a list of task-resource IDs for the specified task:

`GET /rapi/tasks/<task_id>/resources`

To download a binary file for the specified resource:

`GET /rapi/resources/<resource_id>`

Download PCAP for Network Traffic Analysis

Download the PCAP file in the same manner as above for the `task_resource`. The line in the resource list that specifies the PCAP is:

`"resource_magic_magic": "bin:pcap"`

You can open the PCAP in the Security Analytics Platform (which has a Wireshark-like feature) for detailed analysis and artifact extraction or you can use an intrusion detection system (IDS) such as Snort or Suricata to identify network anomalies.

Conditional Processing

Conditional processing is often performed based on the particular results contained within a specific task.

When connected to the `task_complete` WebSocket, a Content Analysis user will receive all task meta-data, including risk score, when the task completes. Alternatively, the user can retrieve a task directly via the API as follows:

`GET /rapi/tasks/<task_id>`

Performing contingent actions programmatically based on the sample's `risk_score` is a common function. Content Analysis maintains two separate `risk_score` values:

`tasks_global_risk_score: 8`	Risk score based on default patterns
`tasks_owner_risk_score: 0`	Risk score based on user's custom patterns

Example

For `risk_score >=7`, write the `task_id` to a log file for additional follow-up analysis.

REST API for File Submission

Symantec provides a REST API for submitting individual files to Content Analysis for evaluation using the current configuration. The API is available to people or programs that want to know how Content Analysis would evaluate a file but don't want to translate it into ICAP, the web-centric protocol that Content Analysis uses. Examples of how the API can be used:

- Use the API with an email gateway to evaluate file attachments.
- Create a script running on a file server to periodically check for malicious files.
- Create a program that submits individual files so that an analyst can see if they are malicious or contain viruses.

This version of the REST API is considered to be phase 1 and is a limited submission API.

To deliver the scanning verdicts to the client, the Pub-Sub API is used. The API is asynchronous and uses WebSocket protocol, which provides full-duplex communication channels over a single TCP connection. The following four-step use case describes how to generate an API key, subscribe to the WebSocket, select the IVM profile to use, and submit files for evaluation.

Sample Python scripts are attached to this PDF. To open the script, double-click the script filename in the Attachments pane on the left.

Step 1: Generate an API Key

The first step to using the REST API for file submission is to generate a key for authenticating to the API. See "Generate an API Key" on page 4 for details.

Other CLI commands are available to view and delete API keys:

- **`ma-actions api-key list`** Shows the ID for each API key and its associated privileges. Note that it does NOT show the value of the key.
- **`ma-actions api-key delete <id>`** Deletes the API key with the specified ID.

Step 2: Subscribe to the WebSocket

An API key allows access to the file submission endpoint and to the WebSocket used for results. It is recommended that you subscribe to the results WebSocket before submitting files so that you don't miss responses. Results for all files submitted via the file submission endpoint are sent to the WebSocket. Each request has a server-generated ID and may contain a client-provided identifier to aid in matching the responses to the appropriate requests.

The URL for the WebSocket is `wss://<CA-host>:<port>/rapi/ws/cas_task`, where *<CA-host>* is the host name or IP address of Content Analysis and *<port>* is the web management HTTPS port. If Content Analysis is configured to use HTTP for web management, you can use `ws://...` instead of `wss:/...`, provided that you specify the HTTP port instead of the HTTPS port.

When attaching to the WebSocket, you need to provide a header with the API key. The header is called "X-API-TOKEN" and its value should be the string obtained from the command line in Step 1.

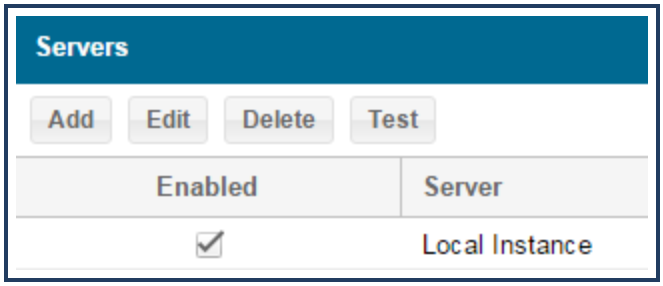
Once attached to the WebSocket, you can listen for messages. Each message is a JSON string representing a Content Analysis result. If sandboxing is enabled on Content Analysis but is configured to not wait for the sandboxing verdict, you

will receive two results for each request that was sent to sandboxing. In other cases, there will be a single response for every request. The format of the JSON response is described in "Appendix A: Asynchronous Response Syntax" on page 30.

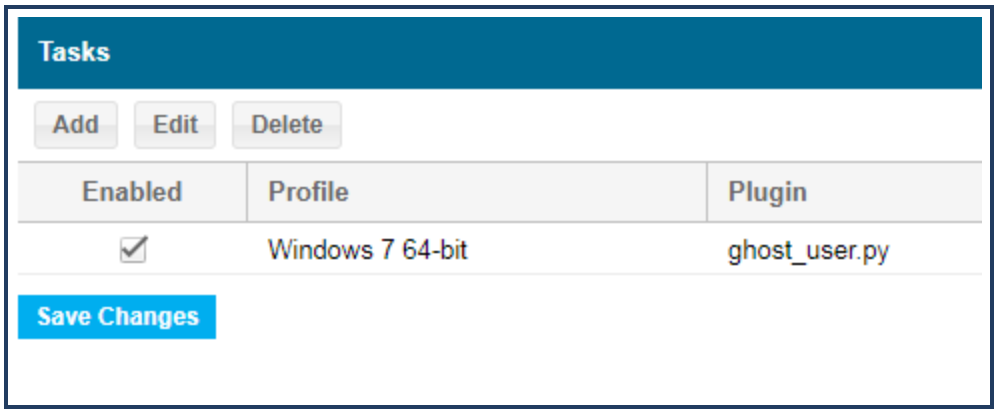
Step 3: Select IVM Profile

Select the IntelliVM profile you want the API to use for files that are submitted for scanning by Content Analysis. In the Content Analysis management console:

1. Select **Services > Sandboxing > Symantec Malware Analysis**.
2. In the Servers panel, make sure **Local Instance** is enabled.



3. In the Tasks panel, specify your profile and plugin configuration.



4. Click **Save Changes**.

Step 4: Submit Files for Evaluation

Once you have an API key, are subscribed to the WebSocket, and have set the default IVM profile to use, you are ready to submit files for evaluation.

The URL for file submission is `https://<server>:<port>/rapi/cas/scan`. *<server>* is the host name or IP address of Content Analysis, and *<port>* is the web management HTTPS port. The discussion in Step 2 about HTTP and HTTPS applies to this URL as well.

When submitting files, you will need to provide a header with the following values:

- **`X-API-TOKEN`**. Its value should be the API key string obtained from the command line in Step 1.
- **`X-Response-Wait-MS`**. The value of this optional setting should be a 32-bit integer that specifies the number of milliseconds for CA to wait before returning the response to the HTTP POST request. If CA receives a verdict before the wait time is reached, it will return the response immediately.

If **`X-Response-Wait-MS`** is set to 0 or not present in the header, all scan results are returned via the WebSocket, and only the server ID is returned as part of the POST.

If you want to specify a client ID, provide it on the query string of the POST request to this URL as a parameter called client-id. For example: POST to `https://<server>:<port>/rapi/cas/scan?client-id=<client specified id>`

The response to the POST request will be a JSON string that indicates whether the response was accepted and, if it was accepted, the server ID assigned to this request. If the verdict was available, the POST response will contain the information and it will not be sent to the WebSocket. It is also possible for a partial verdict to come back via the POST response, and to later be updated via the WebSocket. The full format of these responses is described in "Appendix B: POST Response Syntax" on page 33. Sample JSON responses for successful and failed requests are listed in "Appendix C: Sample JSON" on page 34.

Example cURL submission:

```
curl -X POST --form upload=@test.cmd https://casva:8082/rapi/cas/scan -H "X-API-TOKEN:1d61ebd6a51b435999cd22e0373c41dc"
```

Appendix A: Asynchronous Response Syntax

All responses contain the following elements:

server_time - The current time on the Content Analysis appliance in an extended ISO string

id - The id that was assigned to the request when it was uploaded to the server

client_id - If the client provided an id when it uploaded the request, it will be included here. Otherwise it will be an empty string

exec_time - The elapsed time in milliseconds since this request was uploaded

filename - The name of the file in the request

score - An integer from 0-10, with 0 meaning safe, 10 meaning malicious, and 5 meaning that the file probably safe, that is, it is not known to be either definitely safe or definitely malicious.

status - An integer representing the status of the request

- **0 (IN_PROGRESS)** - This indicates that the response is not the final response and to expect another response when sandboxing is complete
- **1 (COMPLETE)** - This indicates that the response is the final response
- **2 (ERROR)** - This indicates that Content Analysis was unable to fulfill the request
- **3 (COMPLETE_WITH_ERRORS)** - This indicates that some aspect of the request was unsuccessful, but that Content Analysis still created a verdict from the other information it has

expect_sandbox - A Boolean that is true if an additional response with the sandboxing verdict is expected for this request

If file type and size policies didn't trigger an early verdict, the request also contains the following elements:

sha1 - The SHA-1 hash of the file

sha256 - The SHA-256 hash of the file

md5 - The MD5 hash of the file

If the status was ERROR (2), the response also contains the following element:

error - A human-readable string describing the reason for the error

If the file was sent to the File Reputation Service (FRS) the following element is present:

file_reputation

score - An integer from 1 to 10, with 10 indicating a known malicious file, and 1 indicating a known safe file. If the score was not present in the FRS database, this element will not be present

status - The status of the FRS request, using the same enumeration as the global status

If the file was blocked or served because of the custom whitelist or blacklist on Content Analysis, the following element is present:

user_hash_list

score - 0 for allowed by the custom whitelist, 10 for blocked by the custom blacklist

status - The status of the user hash list, using the same enumeration as the global status

If the request was scanned by Cylance, the response includes the following element:

cylance

score - An integer from 0 to 10, with 10 being malicious and 0 being safe.

status - The status of the Cylance request, using the same enumeration as the global status

data_version - The version of the Cylance rules

engine_version - The version of the Cylance binary

details - Details from the Cylance evaluation of the file

If the Content Analysis policy triggered the verdict for the file, the following element is present:

policy

score - 0 for allowed by policy, 10 for blocked by policy

status - The status of the policy, using the same enumeration as the global status

code - The reason code for which policy triggered the verdict. These codes are strings and are the same as the X-Error-Code values returned from ICAP. This field would be useful for automation as the strings will not change

details - A human-readable explanation of which policy triggered the verdict

If AV scanning was run on the file and there were errors or a virus found, one or more of the following elements is present on the response:

kaspersky

sophos

mcafee

symantec

Each of the elements has the same sub-elements:

score - 10 for known bad, 5 for not known bad

status - The status of the antivirus request, using the same enumeration as the global status

engine_version - The version of the binary for the vendor

pattern_version - The version of the patterns in use for the vendor

pattern_date - The date and time of the patterns in use

file_name - The file name that caused the AV verdict

subfile_name - The file within the file that caused the AV verdict

error_code - "virus_found" or any of the X-Error-Code values returned from ICAP. This field is suitable for machine consumption as the strings will not change

error_details - A human-readable string indicating the reason for the verdict

If a virus was found, it also contains the following element:

virus_name - The name assigned to the virus by the AV vendor

If the file was evaluated using a sandbox and found to be a threat, one or more of the following elements is present on the response:

malware_analysis

fireeye

lastline

cloud_sandboxing

Each of these elements has the same sub-elements:

score - For Malware Analysis, an integer from 0 to 10, with 10 indicating a malicious file. FireEye returns either 0 for safe or 1 for malicious. Lastline uses a scale from 0-100 with 100 indicating malicious. Cloud Sandboxing returns a 0 (safe) or 10 (malicious) score.

status - The status of the sandboxing request, using the same enumeration as the global status

report_url - A URL for a detailed report about the sandboxing task

pdf_url - A URL where a detailed report in PDF form can be obtained

Appendix B: POST Response Syntax

The response contains the following elements:

api_version - 1

exec_time - The elapsed time, in milliseconds, that it took to process this request and add the request to the processing queue

server_time - The current time on the Content Analysis appliance in an extended ISO string

request - "POST /rapi/cas/scan"

result

status - The status of the request, either ERROR (0), or IN_PROGRESS (1), using the same enumeration as the global status in the asynchronous responses

If the status is ERROR (0), the following element is present under the result:

error - The reason the request was rejected, currently one of:

"Unauthorized" - Indicates that the API key provided could not be verified

"No file uploaded" - Indicates that the file was not found in the upload

"Multiple files not supported" - Indicates that there were multiple files in the upload

Otherwise, the following element is present under the result:

id - the server-assigned ID for this request (string)

Appendix C: Sample JSON

Sample POST Responses

A response to a successful request:


```
{
     "client_id": '',
     "exec_time": 0.0188,
     "expect_sandbox": False,
     "filename": "PDF_V1_5_MPP.pdf",
     "id": "8d610cdd-1eec-40f5-af63-6e48f463b20b",
     "request": "POST /rapi/cas/scan",
     "score": 5,
     "server_time": "2017-12-29T14:01:20.821277",
     "sha1": "a7d96611b23ad872cbe96c235c1f0b3ea0977655",
     "status": 1
}
```


A response to a failed request


```
{
     "client_id": '',
     "exec_time": 0.0188,
     "expect_sandbox": False,
     "filename": "PDF_V1_5_MPP.pdf",
     "id": "8d610cdd-1eec-40f5-af63-6e48f463b20b",
     "request": "POST /rapi/cas/scan",
     "score": 5,
     "server_time": "2017-12-29T14:01:20.821277",
     "sha1": "a7d96611b23ad872cbe96c235c1f0b3ea0977655",
     "result": {
          "error": "Unauthorized",
          "status": 0
          }
}
```

Sample Asynchronous Responses

A successful response:


```
{
     "server_time": "2016-01-15T18:34:33.941133",
     "id": "8d610cdd-1eec-40f5-af63-6e48f463b20b",
     "client_id": "15f80202-1b0c-4cd9-a467-75cc3d4f26c9",
     "score": 10,
     "exec_time": 0.103,
     "status": 1,
     "sha1": "4660dcd9b6b1f436d7fa202ad1889f6e7fda77d5",
     "sha256":
     "ace4012e8b1789554d2bd8fba106bbee0cb4f088c91ff7f38ea21e810f61299f",
     "md5": "aa7e92df14f21eb6eca314d161c20c52",
     "expect_sandbox": false,
     "file_reputation": {
          "status": 1,
          "score": 8
     },
     "user_hash_list": {
          "status": 1,
          "score": 10
     },
     "cylance": {
          "status": 1,
          "score": 5,
          "data_version": "1235.78",
          "engine_version": "1234.1",
          "details": ""
     },
     "policy": {
          "status": 1,
          "score": 10,
          "code": "blocked_extension",
          "details": "Blocked extension detected: exe"
     },
```

```
        "symantec": {

             "status": 1,
             "score": 5,
             "engine_version": "1.0.1.18",
             "pattern_version": "20170817.186908",
             "pattern_date": "2017/08/18"

        },

        "sophos": {

             "status": 1,
             "score": 10,
             "engine_version": "3.69.3",
             "pattern_version": "5.42",
             "pattern_date": "2017/08/17",
             "virus_name": "Something nasty",
             "file_name": "archive.zip"
             "subfile_name": "dir/something.exe",
             "error_code": 25,
             "error_details": "Virus found in something.exe: Something
             nasty"

        },

        "bcma": {

             "status": 1,
             "score": 5,
             "report_url": "http://ma/report/taskid",
             "pdf_url": "http://ma/report/taskid.pdf"

        },

        "lastline": {

             "status": 1,
             "score": 3,
             "report_url": "http://url",
             "pdf_url": ""

        },

        "FireEye": {

             "status": 1,
             "score": 0
             "report_url": "http://url",
             "pdf_url": ""

        }

    }
```

An error case:


```
{
"server_time": "2016-01-15T18:34:33.941133",
"id": "8d610cdd-1eec-40f5-af63-6e48f463b20b",
"client_id": "15f80202-1b0c-4cd9-a467-75cc3d4f26c9",
"exec_time": 0.133,
"status": 2,
"error": "Bad request"
}
```


A partial success:


```
{
     "server_time": "2016-01-15T18:34:33.941133",
     "id": "8d610cdd-1eec-40f5-af63-6e48f463b20b",
     "client_id": "15f80202-1b0c-4cd9-a467-75cc3d4f26c9",
     "score": 5,
     "exec_time": 0.103,
     "status": 3,
     "sha1": "4660dcd9b6b1f436d7fa202ad1889f6e7fda77d5"
     "sha256":
     "ace4012e8b1789554d2bd8fba106bbee0cb4f088c91ff7f38ea21e810f61299f",
     "md5": "aa7e92df14f21eb6eca314d161c20c52",
     "file_reputation": {
          "status": 1,
          "score": 8
     },
     "user_hash_list": {
          "status": 1,
          "score": 10
     },
     "symantec": {
          "status": 2,
          "score": 5,
          "engine_version": "1.0.1.18",
          "pattern_version": "20170817.186908",
          "pattern_date": "2017/08/16",
          "file_name": "something.exe",
          "subfile_name": "",
          "error_code": "decompression_error",
          "error_details": "failed to decompress archive: 0x80034233"
```

```
        },
        "lastline": {
            "status": 2,
            "error": "Connection error"
        },
        "FireEye": {
            "status": 1,
            "score": 0
        }
    }
```

Appendix D: Example Python Scripts

Example Python Code: WebSocket Task_Complete Notifications

This example demonstrates how to receive basic notifications for completed tasks. Note that only on-box sandboxing results are sent over this WebSocket. It does not include results from antivirus or other Content Analysis scanning technologies.

```
# DISCLAIMER: This code is for API demonstration purposes only.

# It is not indented for production use without modification.

Description: This example demonstrates using WebSockets to receive task_complete
notifications. Note that no authentication is performed in this script, so it will
only work on CA systems where authentication has been disabled.

"""

import sys

import argparse

import json

import ssl

from websocket import create_connection

def parse_message(msg, min_score):

     msg = json.loads(msg)

     score = msg['task']['tasks_global_risk_score']

     if int(score) < min_score:

     return

     sample = msg['sample']

     rsrc = sample['sample_resources'].values()[0]

     magic = rsrc['resource_magic_magic']

     md5 = rsrc['sample_resources_md5']

     print('%s %d %s' % (md5, score, magic))

def listener(server, min_score, key):

     url = "wss://%s/rapi/ws/task_complete/*" % server

     ssl_options = {'cert_reqs': ssl.CERT_NONE} # bypass certificate verification

     ws = create_connection(url, header={'X-API-TOKEN': key}, sslopt=ssl_options)

     while True:

          parse_message(ws.recv(), min_score)
```

```
    ws.close()

def main(server, min_score, key):

    listener(server, min_score, key)

if __name__ == '__main__':

    parser = argparse.ArgumentParser(description='example multiple websocket client')

    parser.add_argument('--key', type=str, required=True, help="Use specified API key
    to authenticate.")

    parser.add_argument('--min-score', type=int, default=0, help='only display tasks
    over this score')

    parser.add_argument('server', type=str, default=['localhost'], help='server
    address')

    args = parser.parse_args()

    sys.exit(main(**vars(args)))
```

Example Python Code: Submit Files to Content Analysis

This example submits files to Content Analysis for evaluation over the REST API. You should first subscribe to the websocket using cas-websocket.py in order to see the responses.

```
#-------------------------------------------------------------------------------
# Dependencies: websocket-client, requests
#
# websocket-client can be downloaded from:
# https://pypi.python.org/pypi/websocket-client
#
# requests can be downloaded from:
# https://codeload.github.com/kennethreitz/requests/legacy.tar.gz/master
#-------------------------------------------------------------------------------
"""
Description: This example submits files to Content Analysis for evaluation over
the REST API. You should first subscribe to the websocket using cas-websocket.py
in order to see the responses.
"""
import sys
import argparse
import json
import ssl
import requests
import os.path
def main(args):
     secure_prefix="s"
     if bool(args.insecure):
          secure_prefix=""
     token = args.key;
     # If no API key is specified, try to acquire one
     if len(token) == 0:
          # Authenticate and get a token
```

```
        auth_url = "http%s://%s/rapi/auth/session" % (secure_prefix, args.host)
        auth_message = { 'username': args.username, 'password': args.password      }
        r = requests.post(auth_url, data=auth_message, verify=False)
        if not r.ok:
             print "failed to authenticate"
             print r
             print r.content
             return -1
        auth = r.json()
        token = auth["results"]["session_token_string"]


    headers = {'X-API-TOKEN': token, 'X-Response-Wait-MS': 1000}
    #CA scan request
    basename = os.path.basename(args.file.name)
    ma_files = { basename: (basename, args.file, 'application/octet-stream') }
    scan_url = "http%s://%s/rapi/cas/scan?token=%s" % (secure_prefix, args.host,
    token)
    r = requests.post(scan_url, files=ma_files, verify=False, headers=headers)
    if not r.ok:
        print "Failed to scan Content Analysis"
        print r
        print r.content
        ws.abort()
        return -1
    print "Success!"
    print r.json()
if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='simple CA websocket example')
    parser.add_argument('-s', '--host', default='localhost', help='CA hostname or IP
    address')
    parser.add_argument('-u', '--username', type=str, required=False,
    default='admin')
```

```
parser.add_argument('-p', '--password', type=str, required=False,
default='admin')

parser.add_argument('-o', '--owner', type=str, required=False, default='admin')

parser.add_argument('-f', '--file', type=argparse.FileType('rb'), required=True)

parser.add_argument('-k', '--key', type=str, required=False, help='The API Key to
use')

parser.add_argument('-i', '--insecure', required=False, default=False,
action='store_true')

sys.exit(main(parser.parse_args()))
```

Example Python Code: WebSocket Scan Notifications

This example demonstrates using websockets to receive scan notifications from the REST API in Content Analysis.

```
#-------------------------------------------------------------------------------
# Dependencies: websocket-client, requests
#
# websocket-client can be downloaded from:
# https://pypi.python.org/pypi/websocket-client
#
# requests can be downloaded from:
# https://codeload.github.com/kennethreitz/requests/legacy.tar.gz/master
#-------------------------------------------------------------------------------
"""
Description: This example demonstrates using websockets to receive scan
notifications from the REST API in Content Analysis. Submit files using
cas-submit.py.
"""
import sys
import argparse
import json
from websocket import WebSocketConnectionClosedException
from websocket import create_connection
import ssl
def websocket_scan_thread(ws):
     while True:
          msg = ""
          try:
               msg = ws.recv()
          except WebSocketConnectionClosedException as e:
               print("Failed to receive: %s" % (e))
               return
```

```
          try:
               print msg
               msg = json.loads(msg)
               #TODO: parse fields out of the json. Right now it just
               # verifies that it is json
          except:
               print("Message in unexpected format: '%s'" % msg)
def main(args):
     secure_prefix="s"
     if bool(args.insecure):
          secure_prefix=""
     token = args.key;
     # If no API key is specified, try to acquire one
     if len(token) == 0:
          # Authenticate and get a token
          auth_url = "http%s://%s/rapi/auth/session" % (secure_prefix, args.host)
          auth_message = { 'username': args.username, 'password': args.password    }
          r = requests.post(auth_url, data=auth_message, verify=False)
          if not r.ok:
               print "failed to authneticate"
               print r
               print r.content
               return -1
          auth = r.json()
          token = auth["results"]["session_token_string"]

     headers = {'X-API-TOKEN': token}
     #subscribe to the websocket
     url = "ws%s://%s/rapi/ws/cas_task" % (secure_prefix, args.host)
     ws = create_connection(url, sslopt={"cert_reqs": ssl.CERT_NONE}, header=headers)
     thread = websocket_scan_thread(ws)
```

```
    thread.start();


if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='simple CA websocket example')
    parser.add_argument('-s', '--host', default='localhost', help='CA hostname or IP
    address')
    parser.add_argument('-u', '--username', type=str, required=False,
    default='admin')
    parser.add_argument('-p', '--password', type=str, required=False,
    default='admin')
    parser.add_argument('-k', '--key', type=str, required=False, help='The API Key to
    use')
    parser.add_argument('-i', '--insecure', required=False, default=False,
    action='store_true')
    sys.exit(main(parser.parse_args()))
```