API reference guide of the CernVM appliance agent

V. Nicolaou[*]

[*]Summer Student 2013, from University of Manchester

1 Introduction

The CernVM appliance agent is a web application that provides useful information of the CernVM and supplies a capability of interactions with it, such as the ability to restart, schedule tasks or do updates. The agent will replace its predecessor created by rPath which is no longer available. For that reason, the new web application aims to be faster, simpler and highly configurable by both users and developers. Thus, an API is provided, in order to add, remove or modify actions that are meant to display information on the screen.

1.1 Structure of the report

This technical report aims to guide in using the API of the CernVM appliance agent to expand its capabilities. The API is formatted in XML and it is eventually a POST request to the IP address that accesses Apache[1] which serves the appliance agent. Each request is analysed separately although requests may have similarities such as the authentication tags. Those tags are discussed under the Request section 2.

1.2 Why use an API?

The functionality and capabilities of a Linux distribution not only varies among each other but it is also changing by itself in an evolutionary way. The appliance agent aims to give users and developers a way of substituting functionality that gets deprecated or adding other functionality that is not in the main release. The project also aims to be useful not only for the Linux distribution running the CernVM but also for any other distribution. Thus, adding extra features may become trivial by supplying a simple API.

1.3 API general information

The API is a set of XML definitions that can be sent via a POST request to the IP address of the appliance agent. The server processes the request and sends back a response, from which the user can figure out if his request was successful or not.

So why XML? XML proves to be both a flexible and portable way of supplying an API since all widely used languages supply libraries for manipulating XML documents. But even if a language does not support such a capability, the XML document can be easily constructed by just assembling strings. Examples of languages are given to cover both examples. An example code in Python is provided.

[1] The Apache Software Foundation, Apache web server, http://www.apache.org/

2 Requests

Before looking at specific requests, the basic properties that are found in every request are mentioned. For a full body request see sections 2.1, 2.2 and 2.3.

- *cernvm-api:* Specifies that the XML being posted is about the CernVM API appliance agent. It also requires a version attribute, currently 1.0
- *username:* The username that manages the appliance agent. At the moment the username is admin.
- *apikey:* The apikey is generated on enabling the API through the appliance agent. It is usually a 13 characters string containing lower letters and numbers.

2.1 The Add Request

The add request is responsible for adding new actions. Modifications of the actions are not allowed, but an action can be removed and another one with the same id can be placed. Listing 1 shows an example of an add request.

Listing 1: Add Request Example code

```
POST https://127.0.0.1:8003/cgi-bin/api/cernvm.py HTTP/1.1
Host: 127.0.0.1:8003
Content-Type: application/xml; charset="utf-8"
Content-Length: 415
Connection: close

<?xml version="1.0" encoding="utf-8"?>
<cernvm-api version="1.0">
  <username>
  admin
  </username>
  <apikey>
  ABCDEF123abcd
  </apikey>
  <add>
      <title>
      Test
      </title>
      <id>
      My Test
      </id>
      <command-group>
        <command title="Hello">
        echo "Hello World"
        </command>
      </command-group>
      <command-group>
        <command title="My hello family table" format="table">
        echo "Message Name"; echo "Hello Styliani";
        echo "Hello Gabriel"; echo "Hello Bill"
        </command>
      </command-group>
  </add>
</cernvm-api>
```

Note the following:

1. The apikey is not the real one. You get the apikey only when you enable the API via the appliance agent.
2. Although the command-group tag has an id attribute in the application, the API does not require it. The system assigns the id automatically.
3. The supplied add request exploits two features of the application. The first is that you can supply more than one command in a single command tag. The second is that there are two formats in which the results of a command are shown in browser. A table format and a raw/simple one.

2.2 The Remove Request

The remove request is simple. It only requires the id of the action that needs to be deleted. Listing 2 shows an example of a remove request.

Listing 2: Remove Request Example code

```
POST https: //127.0.0.1 :8003/cgi-bin/api/cernvm.py HTTP/1.1
Host: 127.0.0.1:8003
Content-Type: application/xml; charset="utf-8"
Content-Length: 415
Connection: close

<?xml version="1.0" encoding="utf-8"?>
<cernvm-api version="1.0">
  <username>
  admin
  </username>
  <apikey>
  ABCDEF123abcd
  </apikey>
  <remove>
      <id>
      My Test
      </id>
  </remove>
</cernvm-api>
```

2.3 The List Request

The list request contains only one single tag, the list. The body of the request is shown on Listing 3. The list request does not do any action on the system. It should be used for information gathering, in order to get the action ids. Thus, the response differs from other requests. It also contains the response id and the response code. If the request was successful (request code = 0) then the request contains also a list of the action ids that are available for modification (i.e. removal). An example of such a response is shown on Listing 4.

Listing 3: List Request Example code

```
POST https: //127.0.0.1 :8003/cgi-bin/api/cernvm.py HTTP/1.1
Host: 127.0.0.1 :8003
Content-Type: application /xml; charset ="utf-8"
Content-Length: 415
Connection: close

<?xml version="1.0" encoding="utf-8"?>
<cernvm-api version="1.0">
  <username>
  admin
  </username>
  <apikey>
  ABCDEF123abcd
  </apikey>
  < list />
</cernvm-api>
```

Listing 4: List Response Example code

```
<?xml version="1.0" encoding="utf-8"?>
<cernvm-api version="1.0">
  <request>
    2387777693429
  </ request >
  <code>
    0
  </code>
  < list >
      <action id="System␣Information"/>
      <action id="Running␣Processes"/>
      <action id="Available␣Packages"/>
  </ list >

</cernvm-api>
```

The list request completes the version 1.0 of the API, which offers simple manipulation on the capabilities of the web appliance agent.

3 Response codes

When a POST request is sent, a response is returned in the form of xml. The code returned upon a response from the API are as shown in Table 1

Table 1: API response codes

Response Code	Meaning
0	Request was successful
10	The API is disabled
22	Unexpected tag found
30	Authentication error/wrong apikey or username
401	Page accessed by browser
500	Internal error when parsing the html/wrong syntax or symantic error

4 Language specific examples

Before doing any API POST requests the API functionality must be enabled. This will generate a key in order to authenticate the POST requests. The API can be enabled under the API Management section. Then check the box 'Enable API' as shown in Figure 1.

When the submit button is pressed, the application generates a key of length 13 with any characters (it may have a fullstop at the end, it's a key character and not a fullstop). Figure 2 shows an example result of enabling API functionality. This result is used in the python specific example in section 4.1. The api-key must be supplied in the application under the tag 'apikey'.

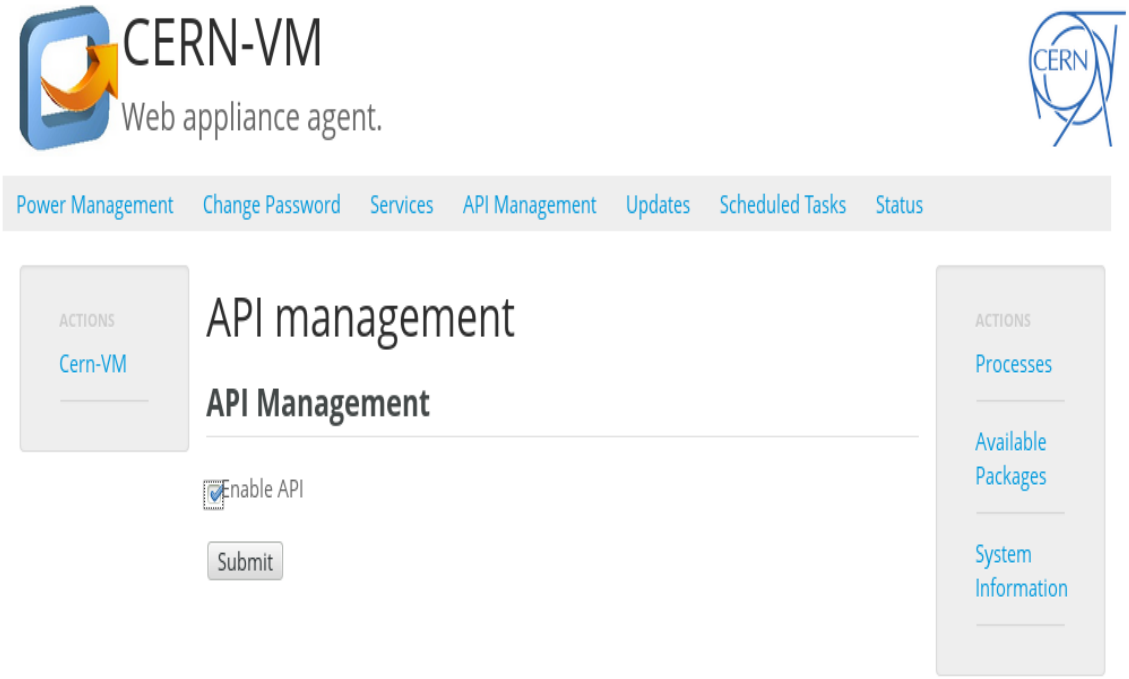

Fig. 1: Click on the box to enable API functionality and then press submit

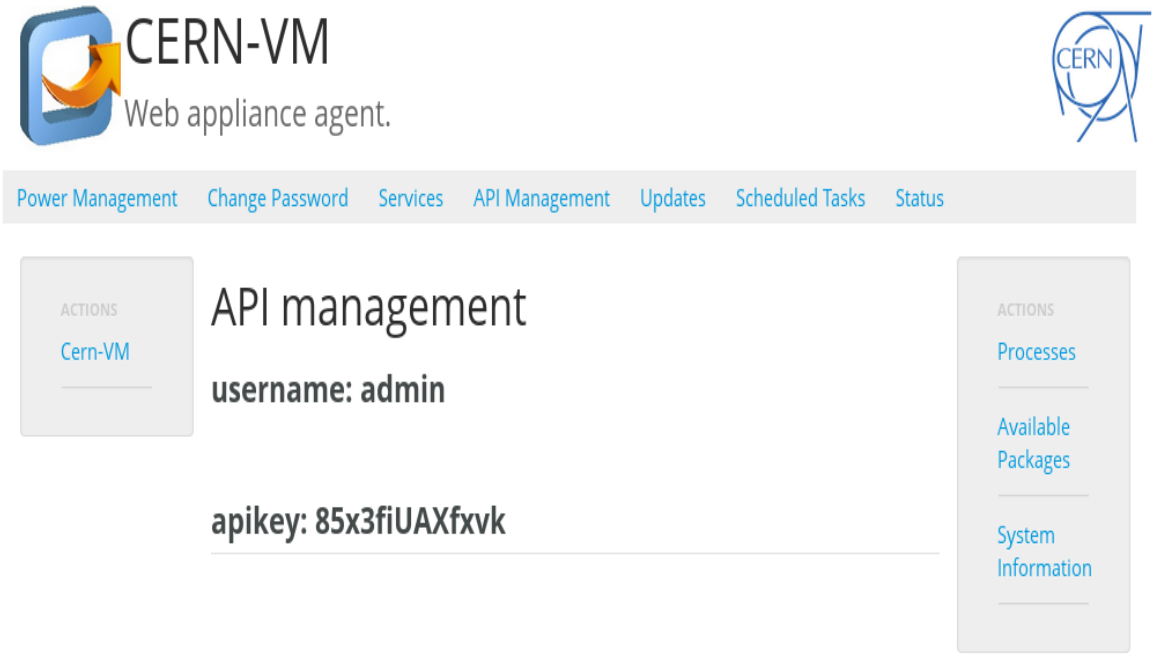

Fig. 2: The username and apikey to be used for in the POST xml document for authenticating the request

4.1 Python

Listing 5: Python Example code

```
#!/usr/bin/python
import xml.etree.ElementTree as ET
import os
import urllib
import urllib2

# testing data
def prepare():
    transaction =ET.Element('cernvm-api', {'version':'1.0'})

    username=ET.Element('username')
    username.text='admin'
    apikey=ET.Element('apikey')
    apikey.text='85x3fiUAXfxvk'
    transaction.append(username)
    transaction.append(apikey)
    return transaction

def test_table(transaction):
    addrequest=ET.Element('add')
    title=ET.Element('title')
    title.text='Hello Family'

    cg=ET.Element('command-group')
    command=ET.Element('command', {'title':'Hello Family', 'format':'table'})
    command.text='echo "Message Name"; echo "Hello Gabriel";' +\
    ' echo "Hello Styliani"; echo "Hello Vasiliki"; echo "Hello George"'

    addrequest.append(title)
    cg.append(command)
    addrequest.append(cg)
    transaction.append(addrequest)
    return transaction

def test_basic(transaction):
    addrequest=ET.Element('add')
    title=ET.Element('title')
    title.text='Hello World'

    cg=ET.Element('command-group')
    command=ET.Element('command', {'title':'Hello'})
    command.text='echo "Hello World"'

    addrequest.append(title)
    cg.append(command)
    addrequest.append(cg)
    transaction.append(addrequest)
```

```
    return  transaction

def post( transaction ):
    xml_string = ET. tostring ( transaction , encoding='UTF-8')
    data = urllib . urlencode({ 'xml': xml_string })
    url=' https: //127.0.0.1 :8003/cgi-bin/api/cernvm.py'
    response = urllib2 . urlopen( url , data )
    for line in response . readlines ():
        print line

transaction =prepare ()
ftr = test_basic ( transaction )
post( ftr )
```