Project 4: Malware Analysis

Instructor: Guanhua Yan

Deadline: December 11, 2020

Project description: In this project the students will need to write a report based on their analysis of one or multiple malware samples. You should consider both static analysis and dynamic analysis of the malware samples chosen. In your report, you should describe what samples you have analyzed, what type of malware they are, what are their likely behaviors, and how to mitigate their attack effects. The report should be at least five pages long, and you can use screenshots taken from your malware analysis.

BigBing Tutorial:

In order to use BigBing for your malware analysis project, you will first need to first register an account through its website: http://cybersec.cs.binghamton.edu/bigbing/index.html. To create an account you need to go to http://cybersec.cs.binghamton.edu/bigbing/python.html and login. Login with your BU id and the password is **cs4558-@user**

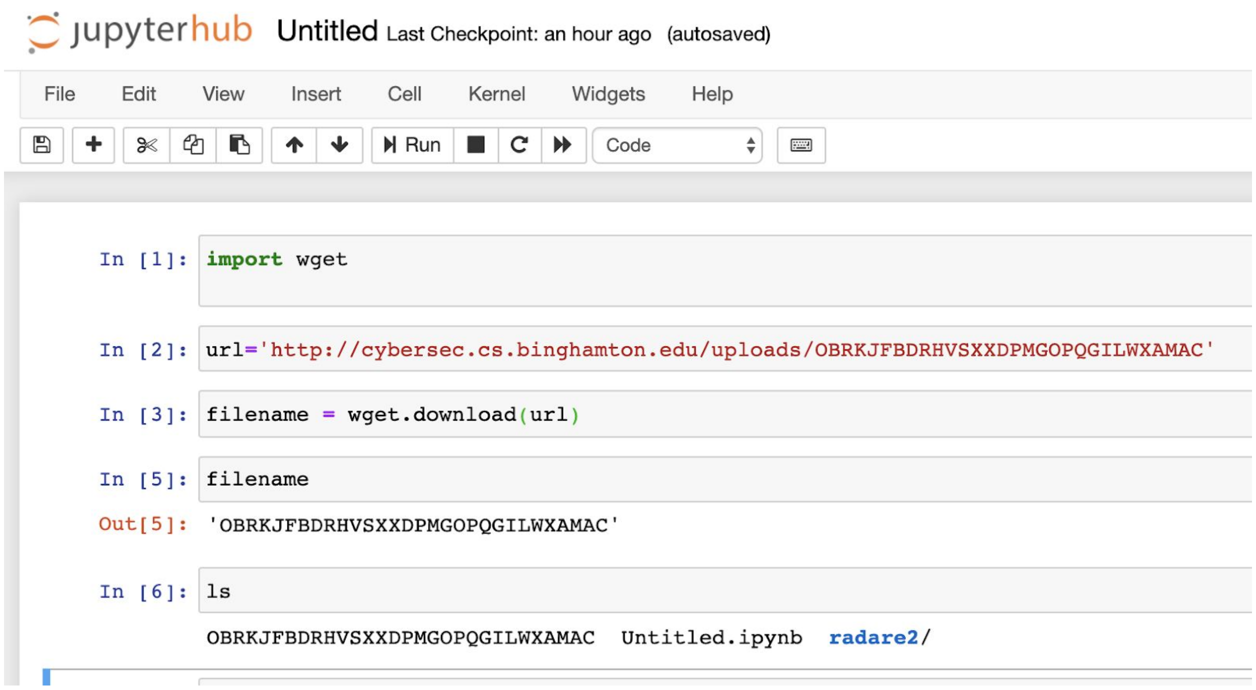
Once you have gotten an account, you can access the BigBing bankend through link: http://puma-4.cs.binghamton.edu:8080/ (this is an internal machine so you would need to log into the campus network through VPN first). In order to connect to the JupyterHub you will need to install Pulse Secure VPN service to connect to the Binghamton University Network. Please follow the instructions on ssl.binghamton.edu

Sample access

You need to download malware samples from BigBing for analysis in this project. To download malware samples, you can take the following steps:

- Go to http://cybersec.cs.binghamton.edu/bigbing/index.html
- Select query section
- You can either search by md5 or filetype & year
- Links will be sent to your email
- Use the wget utility or a browser to download the sample to your directory

It is noted that malware execution can cause damage to information stored on your personal computer. So you should be cautious about downloading the malware samples from BigBing to your local machine. If you would like to perform malware analysis on your own machine, you should use a virtual machine to download the malware samples from BigBing. Otherwise, you should consider using the BigBing platform for malware analysis. Please follow the instructions below about how to use JupiterHub.

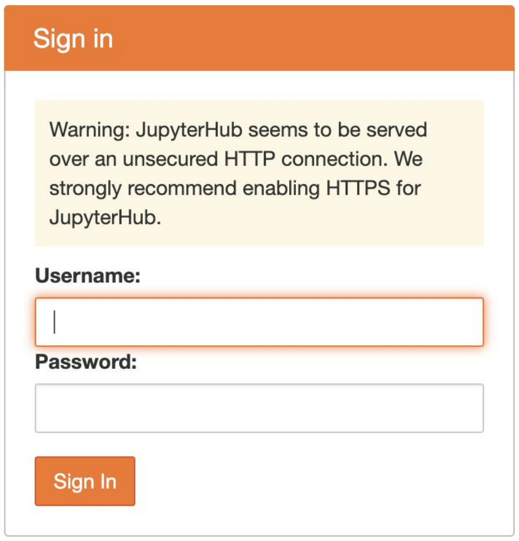

jupyterhub Untitled Last Checkpoint: an hour ago (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Run
Code

```
In [1]: import wget

In [2]: url='http://cybersec.cs.binghamton.edu/uploads/OBRKJFBDRHVSXXDPMGOPQGILWXAMAC'

In [3]: filename = wget.download(url)

In [5]: filename
Out[5]: 'OBRKJFBDRHVSXXDPMGOPQGILWXAMAC'

In [6]: ls
        OBRKJFBDRHVSXXDPMGOPQGILWXAMAC  Untitled.ipynb  radare2/
```

wget script

JupyterHub

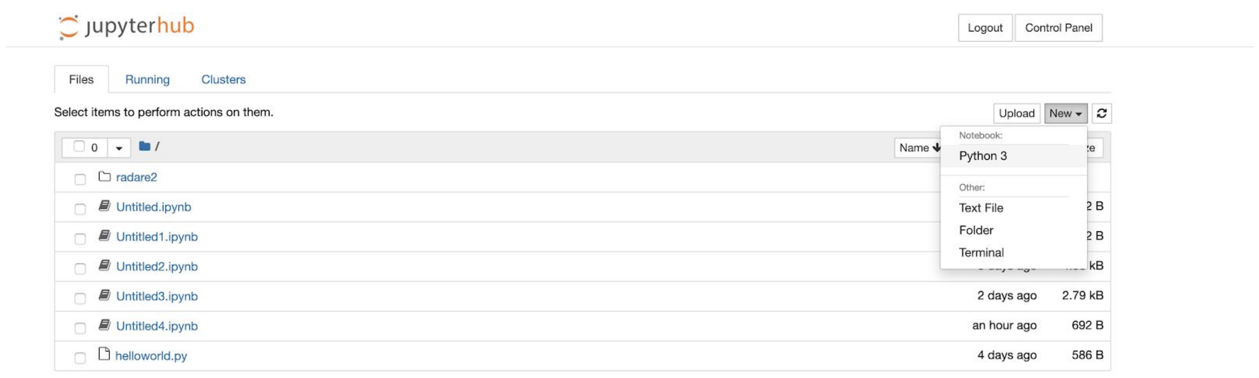
Once your account is created, please go to the BigBing backend: http://puma-4.cs.binghamton.edu:8080/. For the first login authentication you'll need to set up a password by just inputting the password you wish to use to the password textbox. After that point that password will be permanent.



Warning: JupyterHub seems to be served over an unsecured HTTP connection. We strongly recommend enabling HTTPS for JupyterHub.

JupyterHub login page

With JupyterHub you’ll have a shared virtual environment (which runs ubuntu) for your project, which includes some of the common tools for malware analysis. In the figure there are some ipynb files. Those files are Jupyter Notebook which is an Ipython console. We are expecting students to use JupyterHub notebooks for their analysis. For some cases you might need to use the Terminal option too but for most of the cases it shouldn’t be necessary.



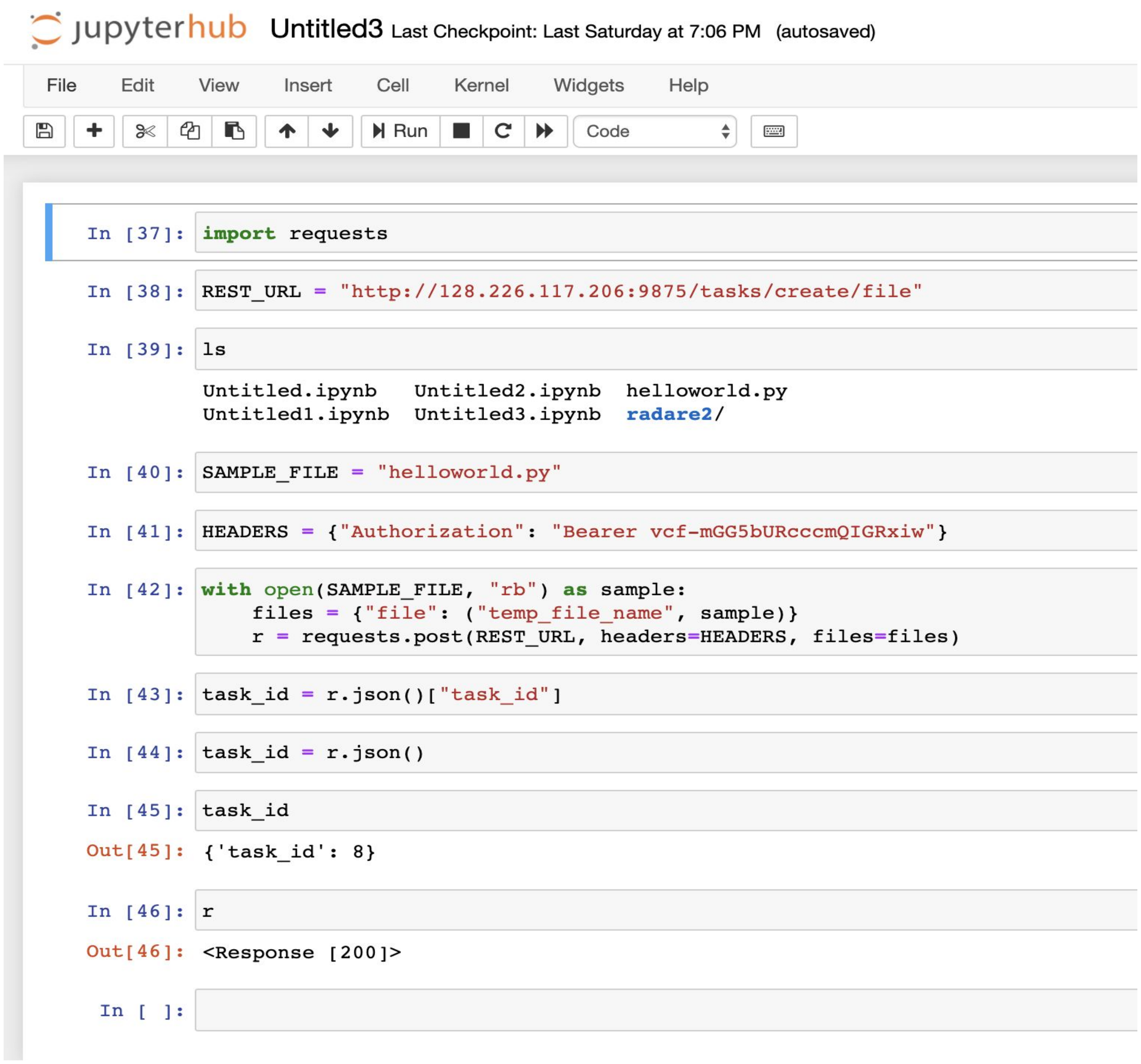
Select items to perform actions on them.

radare2		
Untitled.ipynb		2 B
Untitled1.ipynb		2 B
Untitled2.ipynb		kB
Untitled3.ipynb	2 days ago	2.79 kB
Untitled4.ipynb	an hour ago	692 B
helloworld.py	4 days ago	586 B

JupyterHub file manager

The image is an example usage of a jupyter notebook. The code you write will be executed line by line by using the run button.

In the remainder of this tutorial, we will present the details of the few tools that have already been installed and are accessible within JupyterHub.


jupyterhub
Untitled3 Last Checkpoint: Last Saturday at 7:06 PM (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Run
Code

```
In [37]: import requests

In [38]: REST_URL = "http://128.226.117.206:9875/tasks/create/file"

In [39]: ls
         Untitled.ipynb   Untitled2.ipynb  helloworld.py
         Untitled1.ipynb  Untitled3.ipynb  radare2/

In [40]: SAMPLE_FILE = "helloworld.py"

In [41]: HEADERS = {"Authorization": "Bearer vcf-mGG5bURcccmQIGRxiw"}

In [42]: with open(SAMPLE_FILE, "rb") as sample:
             files = {"file": ("temp_file_name", sample)}
             r = requests.post(REST_URL, headers=HEADERS, files=files)

In [43]: task_id = r.json()["task_id"]

In [44]: task_id = r.json()

In [45]: task_id
Out[45]: {'task_id': 8}

In [46]: r
Out[46]: <Response [200]>

In [ ]:
```

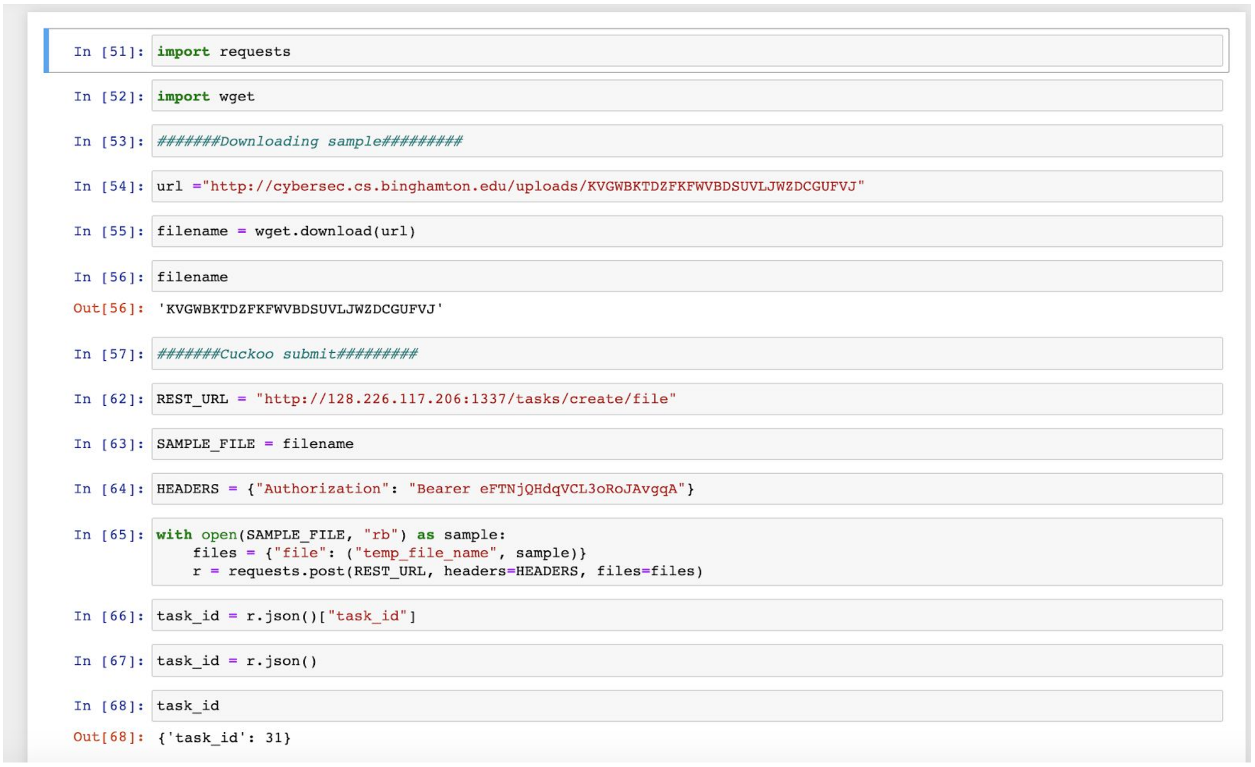
Example Jupyter Notebook

<u>Cuckoo Sandbox</u> 主要使用这个写报告

You will use a rest api in order to submit samples and get results back. We also provided a web interface for you but we encourage students to use rest api. You'll be sending http requests to our cuckoo sandbox which runs on one of our nodes. Information about how to use the Cuckoo sandbox can be found here: https://cuckoo.readthedocs.io/en/latest/usage/api/

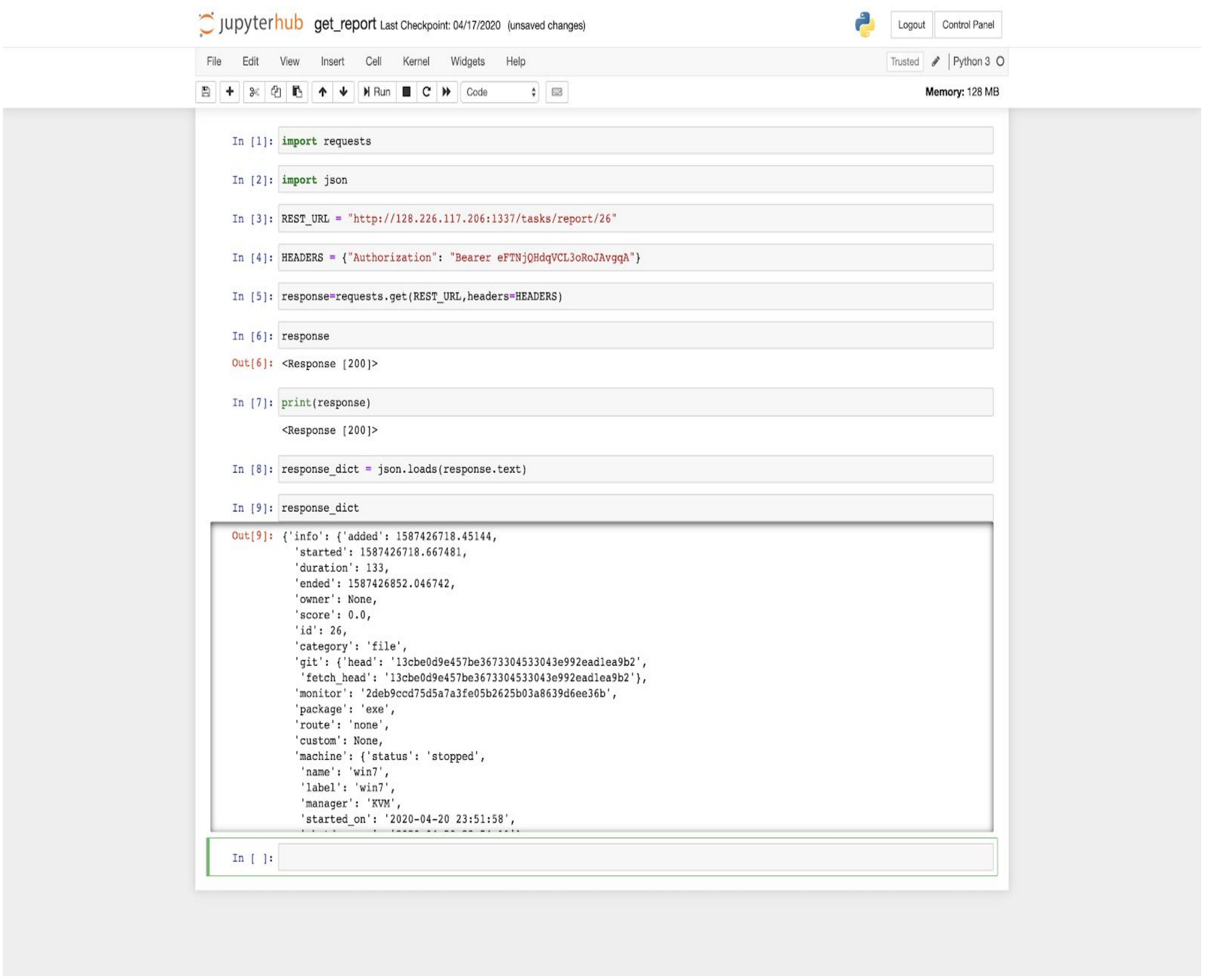
The Cuckoo web interface can be accessed from http://puma-4.cs.binghamton.edu:9876/
API:http://128.226.117.206:1337/
API Authorization: Bearer eFTNjQHdqVCL3oRoJAvgqA

Sample scripts



```
In [51]: import requests

In [52]: import wget

In [53]: #######Downloading sample#########

In [54]: url ="http://cybersec.cs.binghamton.edu/uploads/KVGWBKTDZFKFWVBDSUVLJWZDCGUFVJ"

In [55]: filename = wget.download(url)

In [56]: filename
Out[56]: 'KVGWBKTDZFKFWVBDSUVLJWZDCGUFVJ'

In [57]: #######Cuckoo submit#########

In [62]: REST_URL = "http://128.226.117.206:1337/tasks/create/file"

In [63]: SAMPLE_FILE = filename

In [64]: HEADERS = {"Authorization": "Bearer eFTNjQHdqVCL3oRoJAvgqA"}

In [65]: with open(SAMPLE_FILE, "rb") as sample:
             files = {"file": ("temp_file_name", sample)}
             r = requests.post(REST_URL, headers=HEADERS, files=files)

In [66]: task_id = r.json()["task_id"]

In [67]: task_id = r.json()

In [68]: task_id
Out[68]: {'task_id': 31}
```

Sample download and submit script


jupyterhub get_report Last Checkpoint: 04/17/2020 (unsaved changes)
Logout
Control Panel
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
Memory: 128 MB

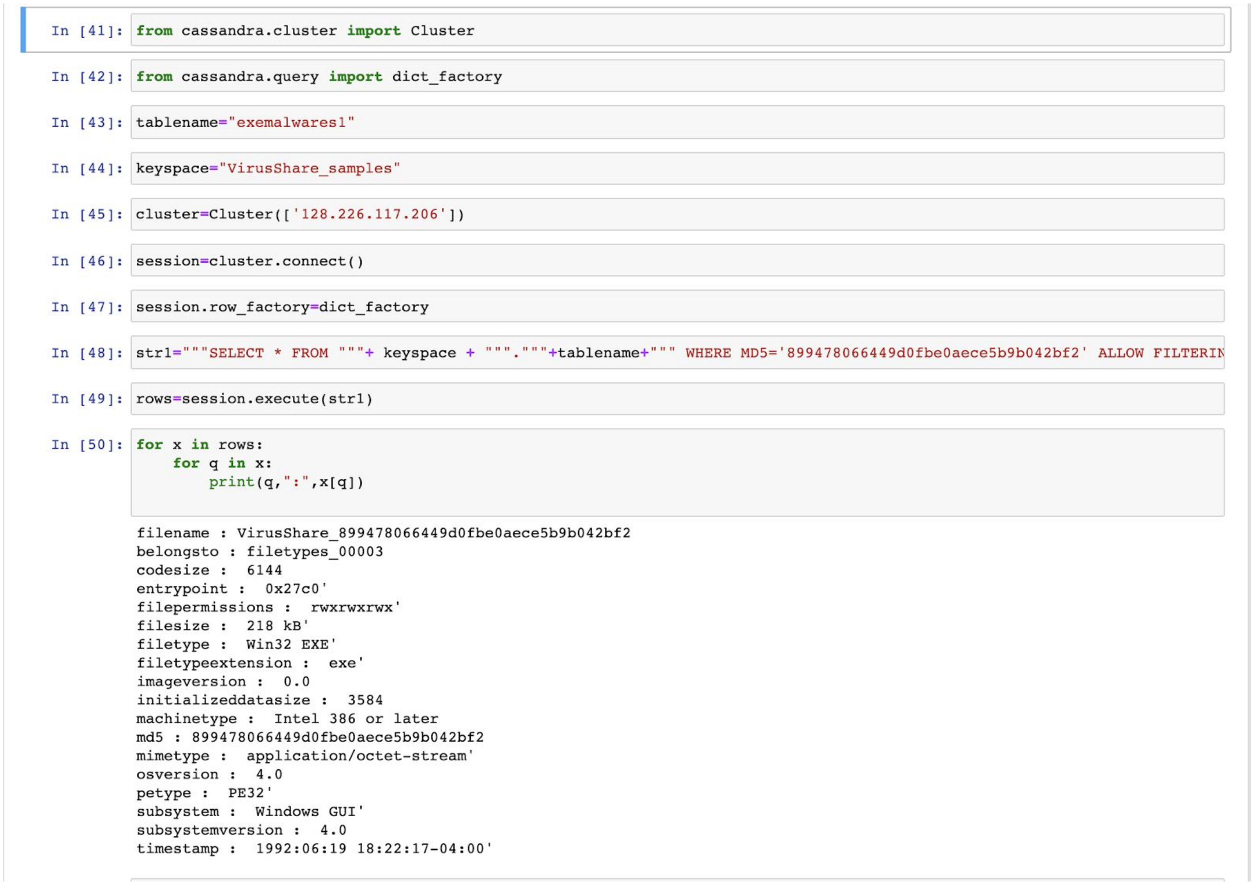
```
In [1]: import requests

In [2]: import json

In [3]: REST_URL = "http://128.226.117.206:1337/tasks/report/26"

In [4]: HEADERS = {"Authorization": "Bearer eFTNjQHdqVCL3oRoJAvgqA"}

In [5]: response=requests.get(REST_URL,headers=HEADERS)

In [6]: response
Out[6]: <Response [200]>

In [7]: print(response)
        <Response [200]>

In [8]: response_dict = json.loads(response.text)

In [9]: response_dict
Out[9]: {'info': {'added': 1587426718.45144,
          'started': 1587426718.667481,
          'duration': 133,
          'ended': 1587426852.046742,
          'owner': None,
          'score': 0.0,
          'id': 26,
          'category': 'file',
          'git': {'head': '13cbe0d9e457be3673304533043e992ead1ea9b2',
           'fetch_head': '13cbe0d9e457be3673304533043e992ead1ea9b2'},
          'monitor': '2deb9ccd75d5a7a3fe05b2625b03a8639d6ee36b',
          'package': 'exe',
          'route': 'none',
          'custom': None,
          'machine': {'status': 'stopped',
           'name': 'win7',
           'label': 'win7',
           'manager': 'KVM',
           'started_on': '2020-04-20 23:51:58',

In [ ]:
```

Cuckoo get view script

If you want to get the analysis result, use "http://128.226.117.206:1337/tasks/report/26" and the others remain the same. Also note that 26 is the task id. If you submit a new malware sample, you may get a different task id, so the URL should be revised accordingly.

Cassandra

We store our malware metadata in a distributed cassandra database. And we arrange the accesses so each student can make queries to our database using the cassandra python driver. For our database the keyspace is "VirusShare_samples" and there are several tables in that keyspace. The important ones that students can use are "doc_table1", "exe_malware1", "saved_samples3". In addition to that while making a query please use the allow filtering option in order to query by a specific column.


```
In [41]: from cassandra.cluster import Cluster

In [42]: from cassandra.query import dict_factory

In [43]: tablename="exemalwares1"

In [44]: keyspace="VirusShare_samples"

In [45]: cluster=Cluster(['128.226.117.206'])

In [46]: session=cluster.connect()

In [47]: session.row_factory=dict_factory

In [48]: str1="""SELECT * FROM """+ keyspace + """."""+tablename+""" WHERE MD5='899478066449d0fbe0aece5b9b042bf2' ALLOW FILTERIN

In [49]: rows=session.execute(str1)

In [50]: for x in rows:
             for q in x:
                 print(q,":",x[q])

         filename : VirusShare_899478066449d0fbe0aece5b9b042bf2
         belongsto : filetypes_00003
         codesize :  6144
         entrypoint :  0x27c0'
         filepermissions :  rwxrwxrwx'
         filesize :  218 kB'
         filetype :  Win32 EXE'
         filetypeextension :  exe'
         imageversion :  0.0
         initializeddatasize :  3584
         machinetype :  Intel 386 or later
         md5 : 899478066449d0fbe0aece5b9b042bf2
         mimetype :  application/octet-stream'
         osversion :  4.0
         petype :  PE32'
         subsystem :  Windows GUI'
         subsystemversion :  4.0
         timestamp :  1992:06:19 18:22:17-04:00'
```

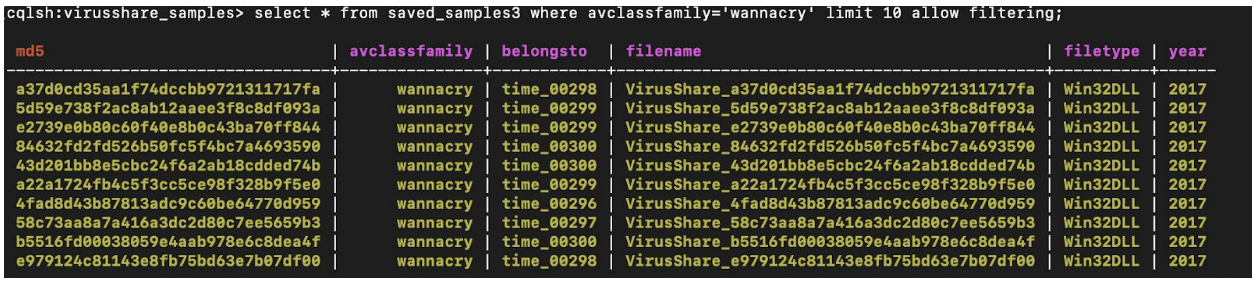
Cassandra example

Malware samples information are currently stored in a cassandra database. Queries can be made by family name, md5, filetype,year, filename, VirusShare folder that it belongs to.

How to Query

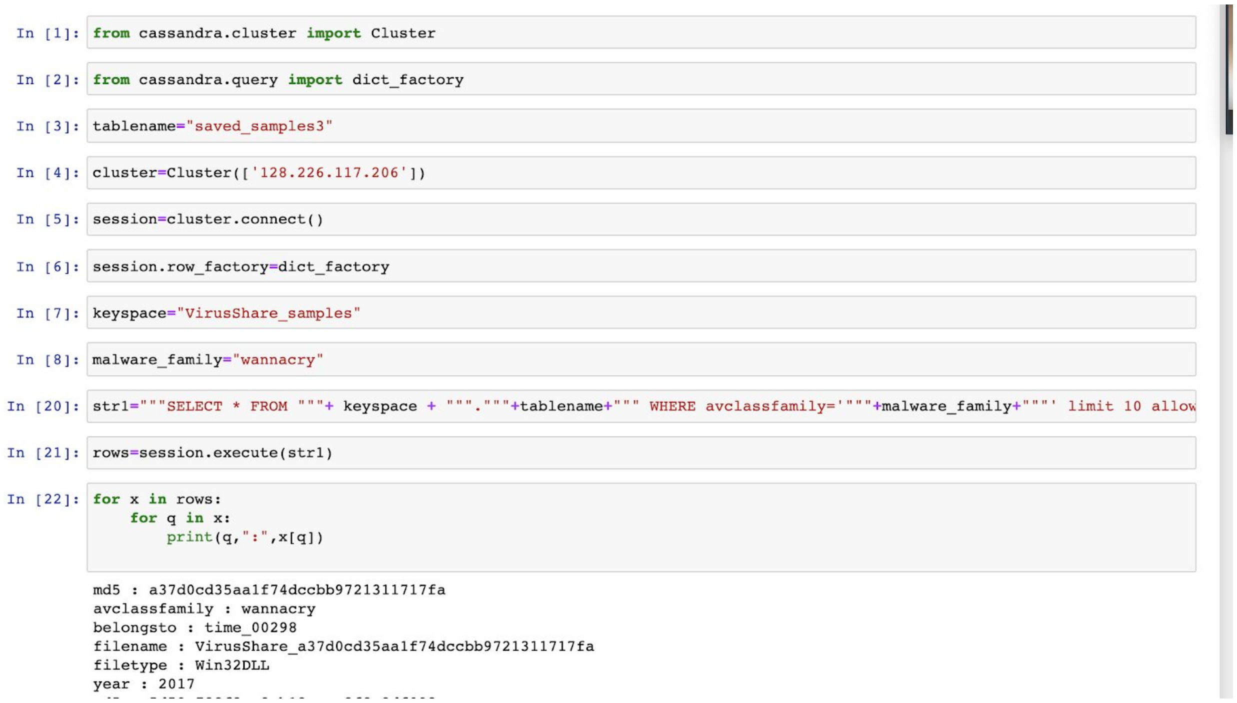
Example query:

Select * from virusshare_samples.saved_samples3 where avclassfamily='wannacry' limit 10 allow filtering;
Primary key on this table is md5 so in order to query by the other columns I used the allow filtering option and the query above will list the first 10 rows which belong to the wannacry family. In addition to the malware family users can make queries to get the entropy of the samples and to figure out which packer is used if the sample is packed(currently only samples packed by upx are shown but the other ones will be added soon.)



```
cqlsh:virusshare_samples> select * from saved_samples3 where avclassfamily='wannacry' limit 10 allow filtering;

 md5                              | avclassfamily | belongsto  | filename                                    | filetype | year
----------------------------------+---------------+------------+---------------------------------------------+----------+------
 a37d0cd35aa1f74dccbb9721311717fa |      wannacry | time_00298 | VirusShare_a37d0cd35aa1f74dccbb9721311717fa | Win32DLL | 2017
 5d59e738f2ac8ab12aaee3f8c8df093a |      wannacry | time_00299 | VirusShare_5d59e738f2ac8ab12aaee3f8c8df093a | Win32DLL | 2017
 e2739e0b80c60f40e8b0c43ba70ff844 |      wannacry | time_00299 | VirusShare_e2739e0b80c60f40e8b0c43ba70ff844 | Win32DLL | 2017
 84632fd2fd526b50fc5f4bc7a4693590 |      wannacry | time_00300 | VirusShare_84632fd2fd526b50fc5f4bc7a4693590 | Win32DLL | 2017
 43d201bb8e5cbc24f6a2ab18cdded74b |      wannacry | time_00300 | VirusShare_43d201bb8e5cbc24f6a2ab18cdded74b | Win32DLL | 2017
 a22a1724fb4c5f3cc5ce98f328b9f5e0 |      wannacry | time_00299 | VirusShare_a22a1724fb4c5f3cc5ce98f328b9f5e0 | Win32DLL | 2017
 4fad8d43b87813adc9c60be64770d959 |      wannacry | time_00296 | VirusShare_4fad8d43b87813adc9c60be64770d959 | Win32DLL | 2017
 58c73aa8a7a416a3dc2d80c7ee5659b3 |      wannacry | time_00297 | VirusShare_58c73aa8a7a416a3dc2d80c7ee5659b3 | Win32DLL | 2017
 b5516fd00038059e4aab978e6c8dea4f |      wannacry | time_00300 | VirusShare_b5516fd00038059e4aab978e6c8dea4f | Win32DLL | 2017
 e979124c81143e8fb75bd63e7b07df00 |      wannacry | time_00298 | VirusShare_e979124c81143e8fb75bd63e7b07df00 | Win32DLL | 2017
```

Result of the query above

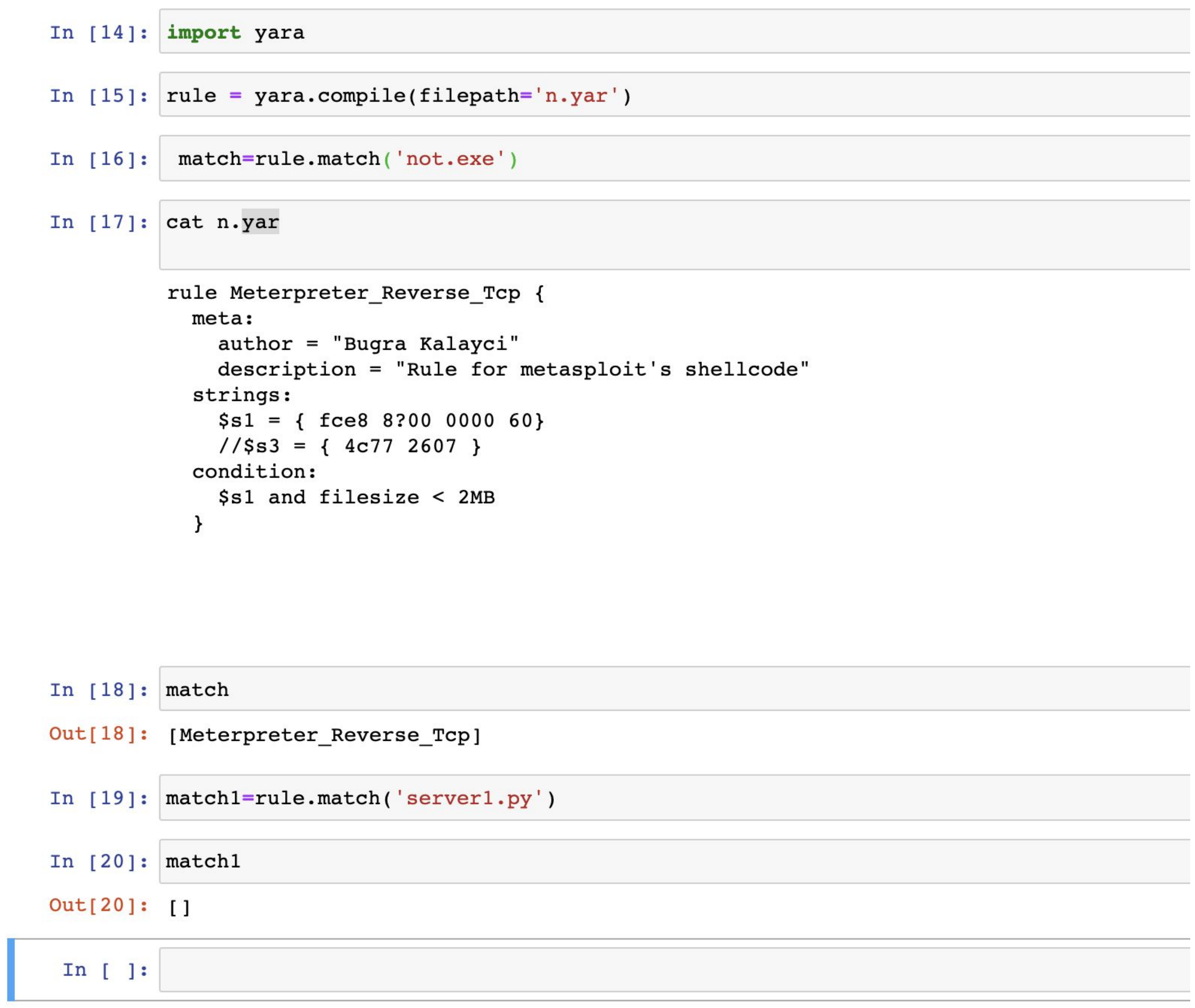


```
In [1]: from cassandra.cluster import Cluster

In [2]: from cassandra.query import dict_factory

In [3]: tablename="saved_samples3"

In [4]: cluster=Cluster(['128.226.117.206'])

In [5]: session=cluster.connect()

In [6]: session.row_factory=dict_factory

In [7]: keyspace="VirusShare_samples"

In [8]: malware_family="wannacry"

In [20]: str1="""SELECT * FROM """+ keyspace + """."""+tablename+""" WHERE avclassfamily='"""+malware_family+"""' limit 10 allow

In [21]: rows=session.execute(str1)

In [22]: for x in rows:
             for q in x:
                 print(q,":",x[q])

         md5 : a37d0cd35aa1f74dccbb9721311717fa
         avclassfamily : wannacry
         belongsto : time_00298
         filename : VirusShare_a37d0cd35aa1f74dccbb9721311717fa
         filetype : Win32DLL
         year : 2017
```

Script for query with malware family

Yara-python

Yara is a malware classification and pattern matching tool. It helps users to detect textual or binary patterns in an executable or a file. In order to detect students need to write some yara rules for their files. The example below is for an executable(“*not.exe”)* which has reverse_tcp_meterpreter payload in it and s1 is for catching the meterpreter session. Documentation links are available if needed.



```
In [14]: import yara

In [15]: rule = yara.compile(filepath='n.yar')

In [16]:  match=rule.match('not.exe')

In [17]: cat n.yar

rule Meterpreter_Reverse_Tcp {
  meta:
    author = "Bugra Kalayci"
    description = "Rule for metasploit's shellcode"
  strings:
    $s1 = { fce8 8?00 0000 60}
    //$s3 = { 4c77 2607 }
  condition:
    $s1 and filesize < 2MB
  }

In [18]: match
Out[18]: [Meterpreter_Reverse_Tcp]

In [19]: match1=rule.match('server1.py')

In [20]: match1
Out[20]: []

In [ ]:
```

Please check the following links:

https://github.com/VirusTotal/yara-python
https://github.com/VirusTotal/yara
https://yara.readthedocs.io/en/v3.4.0/yarapython.html

Yara also provided example rules for different kinds of malwares and viruses. Please check the link below.

https://github.com/Yara-Rules/rules

Radare2

Reverse engineering tool & disassembler. Information about the radare2 tool can be found at the following link:

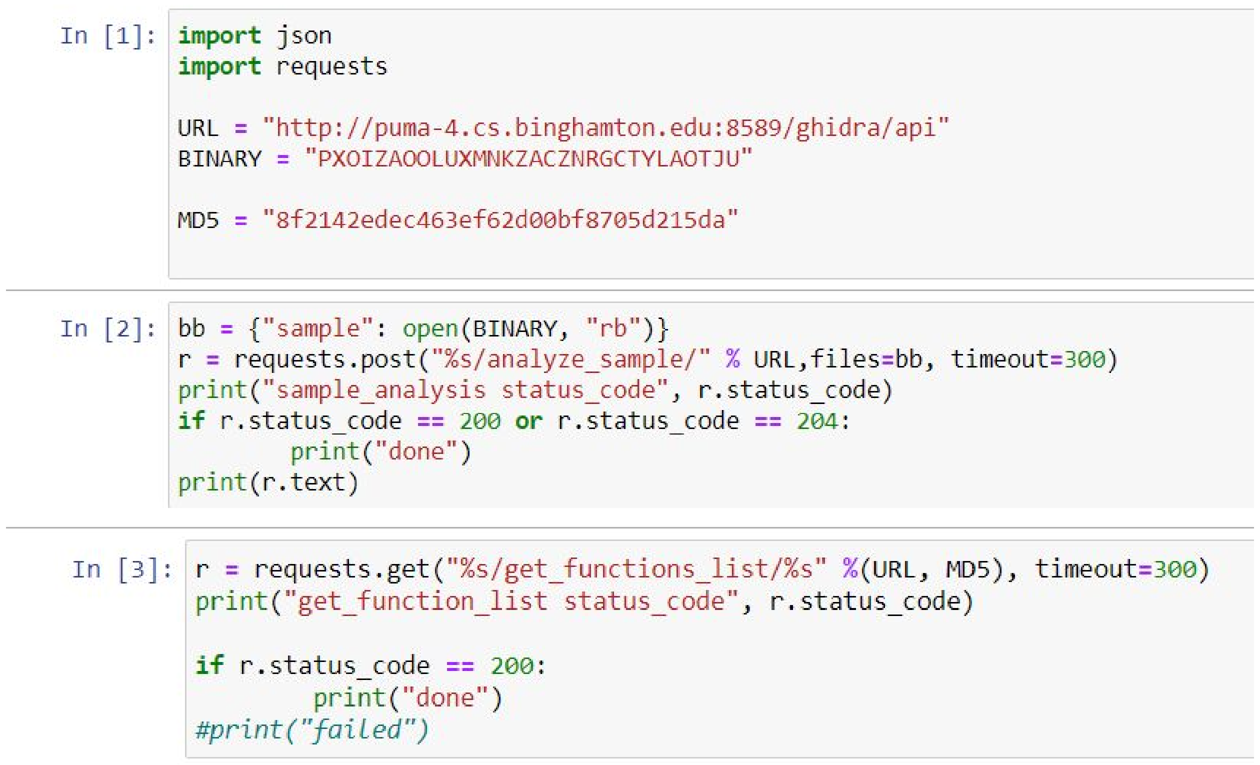
https://rada.re/n/radare2.html

Ghidra

Ghidra is a software reverse engineering (SRE) suite of tools developed by NSA. It is especially useful for decompiling parts of or the entire malware.
Ghidra usage is pretty similar to how we use cuckoo.

You will require the below information -
URL = "http://puma-4.cs.binghamton.edu:8589/ghidra/api"
Filename - The randomly generated string (PXOIZAOOLUXMNKZACZNRGCTYLAOTJU in our example)
MD5 of the sample - Provided when you obtain the sample and also y most of the analysis tools.

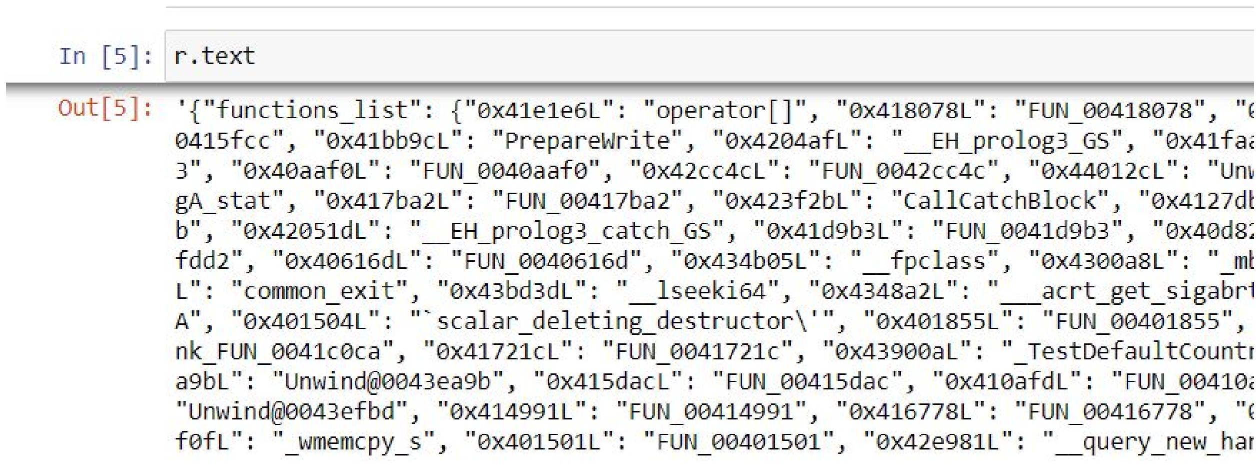
Below is a complete example for obtaining the functions list-



```
In [1]: import json
        import requests

        URL = "http://puma-4.cs.binghamton.edu:8589/ghidra/api"
        BINARY = "PXOIZAOOLUXMNKZACZNRGCTYLAOTJU"

        MD5 = "8f2142edec463ef62d00bf8705d215da"
```

```
In [2]: bb = {"sample": open(BINARY, "rb")}
        r = requests.post("%s/analyze_sample/" % URL,files=bb, timeout=300)
        print("sample_analysis status_code", r.status_code)
        if r.status_code == 200 or r.status_code == 204:
                print("done")
        print(r.text)
```

```
In [3]: r = requests.get("%s/get_functions_list/%s" %(URL, MD5), timeout=300)
        print("get_function_list status_code", r.status_code)

        if r.status_code == 200:
                print("done")
        #print("failed")
```



```
In [5]: r.text

Out[5]: '{"functions_list": {"0x41e1e6L": "operator[]", "0x418078L": "FUN_00418078", "
        0415fcc", "0x41bb9cL": "PrepareWrite", "0x4204afL": "__EH_prolog3_GS", "0x41fa
        3", "0x40aaf0L": "FUN_0040aaf0", "0x42cc4cL": "FUN_0042cc4c", "0x44012cL": "Unw
        gA_stat", "0x417ba2L": "FUN_00417ba2", "0x423f2bL": "CallCatchBlock", "0x4127d
        b", "0x42051dL": "__EH_prolog3_catch_GS", "0x41d9b3L": "FUN_0041d9b3", "0x40d8
        fdd2", "0x40616dL": "FUN_0040616d", "0x434b05L": "__fpclass", "0x4300a8L": "_m
        L": "common_exit", "0x43bd3dL": "__lseeki64", "0x4348a2L": "___acrt_get_sigabr
        A", "0x401504L": "`scalar_deleting_destructor\'", "0x401855L": "FUN_00401855",
        nk_FUN_0041c0ca", "0x41721cL": "FUN_0041721c", "0x43900aL": "_TestDefaultCount
        a9bL": "Unwind@0043ea9b", "0x415dacL": "FUN_00415dac", "0x410afdL": "FUN_00410
        "Unwind@0043efbd", "0x414991L": "FUN_00414991", "0x416778L": "FUN_00416778", "
        f0fL": "_wmemcpy_s", "0x401501L": "FUN_00401501", "0x42e981L": "__query_new_ha
```

You must **<u>always</u>** first analyze the sample and then you can perform the following functions-

- Get decompiled function - /ghidra/api/get_decompiled_function/<MD5>/<function_offset>
- Get functions list - /ghidra/api/get_functions_list/<MD5>
- Get detailed functions list - /ghidra/api/get_functions_list_detailed/<MD5>

Remember, if you strictly follow the same python code as in the example above, you will need to make changes in the requests.get or requests.post methods & build URLs properly.

Let us now if you need to use any other tool. We will add more functionalities if needed. For the lecture we will show some scripts and we'll share those scripts after the tutorial.

<u>References</u>

https://ssl.binghamton.edu/
http://cybersec.cs.binghamton.edu/bigbing/index.html
https://github.com/VirusTotal/yara-python
https://github.com/VirusTotal/yara
https://virustotal.github.io/yara/
https://cuckoo.readthedocs.io/en/latest/usage/api/
https://rada.re/n/
https://docs.datastax.com/en/developer/python-driver/3.22/api/cassandra/cluster/
https://www.megabeets.net/a-journey-into-radare-2-part-1/
https://rada.re/n/radare2.html
https://github.com/Yara-Rules/rules
https://yara.readthedocs.io/en/v3.4.0/yarapython.html
https://ghidra-sre.org/