**TECHNICAL DOCUMENTATION**

**for**

**A Web Scraping of Attack.mitre website**

**PROJECT**

**By**

**Rohit Dhaipule**

**Table of Contents**

[1. Getting Started 3](#_Toc43566917)

[2. Extraction of links 4](#_Toc43566918)

[3. Extraction of Detections, keywords and Mitigations 6](#_Toc43566919)

[4. Creating a webpage for displaying the acquired data 10](#_Toc43566920)

The main aim of the project is to extract all the links from a website and also to extract data from website using python. This project has three parts.

1. Extracting all the links from a given website.
2. Extracting the detections, their keywords and the related mitigations from all the links in the techniques section of the website - <http://attack.mitre.org/> .
3. Displaying all the extracted information on a webpage using Django framework.

# Getting Started

If you don’t have python3 installed in your pc then, install python3. You can refer to python documentation for installation steps. ( <https://docs.python.org/3/> ). Make sure python works from command prompt

Then install the following python libraries. The commands for installing them are given below. Run the commands in the command prompt to install the corresponding libraries.

* re library

*pip install regex*

* beautifulsoup library

*pip install beautifulsoup4*

* rake-nltk library

*pip install rake-nltk*

* xlsx-writer library

*pip install XlsxWriter*

You can refer to the documentations of those libraries for more information.

Now install Django framework in your system. It can be done by the command:

*pip install Django*

For further queries refer to Django documentation. (<https://www.djangoproject.com/download/> )

Now installation of following Django libraries is to be completed. The commands for installing them are given below. Run the commands in the command prompt to install the corresponding libraries.

* django pagination library

*pip install django-pagination*

* django import-export library

*pip install django-import-export*

# Extraction of links

For this part, first a url from queue is parsed and all the html content is extracted using beautifulsoup. Then the content of href attribute of all the anchor tags are saved in a queue. Then the crawled link is moved to a crawl (a queue which stores all the crawled links). If due to some external factors the file has to stop running then when we run the next time it starts running from the same point where it was stopped. Add first to a queue file and then to a crawl file makes this possible.

The following functions are used in extraction of links.

* def set\_to\_file (links, file\_name)

This function helps to write all the content of a set to a file. It takes file name and a set as input and loops through all the contents of the set and add them to the file.

def set\_to\_file (links, file\_name):

with open(file\_name,"w") as f:

for l in sorted(links):

f.write(l+"\n")

* def file\_to\_set (file\_name)

This function helps to write all the content of a file to a set. It takes file name as input and loops through all the contents and add them to a set ‘results’ and returns it.

def file\_to\_set(file\_name):

results = set()

with open(file\_name, 'rt') as f:

for line in f:

results.add(line.replace('\n', ''))

return results

* def get\_domain\_name (url, remove\_http=True)

This function returns the domain name of a given website link. This uses a function namely urlparse which parses a URL into six components, returning a 6-tuple. Each tuple item is a string, possibly empty. The netloc returns the domain name of the website.

def get\_domain\_name (url, remove\_http=True):

uri = urlparse(url)

if remove\_http:

domain\_name = f"{uri.netloc}"

else:

domain\_name = f"{uri.netloc}://{uri.netloc}"

return domain\_name

* def get\_links (link, domain\_name, queue, crawl)

This function takes a link, domain name, queue and crawl as input extracts all the links in the given link which are not in queue or crawl files. This also filters the links which doesn’t have same domain name as the given link. This is done to avoid crawling out of range (to avoid crawling of whole twitter, google etc).

The first 4 lines of the function opens the links and encodes it to UTF-8 encoding. The first 3 lines are to avoid authentication of certificates. The next line uses beautifulsoup returns the html content of a link. All the anchor tags with href attributes are represented by tags. All the other steps are self-explanatory.

def get\_links (link, domain\_name, queue, crawl):

try:

ctx = ssl.create\_default\_context()

ctx.check\_hostname = False

ctx.verify\_mode = ssl.CERT\_NONE

url = urllib.request.urlopen(link, context=ctx).read()

soup = BeautifulSoup(url,'html.parser')

tags = soup('a',href=True)

for tag in tags:

href = tag['href']

#To avoid links without domain name in it

if href.startswith('/'):

href = 'https://' + domain\_name + href

#To avoid replication of links in queue or crawl lists

if href.endswith('/'):

href = href[:-1]

if get\_domain\_name(href) == domain\_name:

if href not in queue:

if href not in crawl :

queue.insert(0,href)

else:

continue

except Exception as e:

print(str(e))

Now moving to the application of these functions for extraction of links.

import re

from general import \*

f = open('url.txt','r')

line = f.readline()

fline = re.findall(r'\S+',line)

if fline[-1].startswith(':'):

homepage = fline[-1][1:]

else:

homepage = fline[-1]

domain\_name = get\_domain\_name(homepage)

queue = list(file\_to\_set('queue.txt'))

crawl = list(file\_to\_set('crawl.txt'))

if (len(queue) == 0) and (len(crawl) == 0):

queue.insert(0,homepage)

while len(queue) > 0:

link = queue[0]

print('Working on ', ' url --- ', link)

if link.endswith('.pdf') or link.endswith('.json') or link.endswith('.png') or link.endswith('.jpg') or link.endswith('.xlsx'):

queue.remove(link)

crawl.insert(0,link)

else:

get\_links(link,domain\_name,queue,crawl)

queue.remove(link)

crawl.insert(0,link)

set\_to\_file(queue, 'queue.txt')

set\_to\_file (crawl, 'crawl.txt')

In this code the of the website is taken as input from a file named ‘url.txt’. Then all the content of the files ‘queue.txt’ and ‘crawl.txt’ are added to queues crawl and queue. If the queues are empty then the website link is added to the queue else the crawling (which has already started) continues. The following lines loop through the queue and extract all links of links from it, then saving it in the corresponding files.

# Extraction of Detections, keywords and Mitigations

For this part, we extract the div tags with id “detection” or “detectable” from html content of all the links in techniques page of <http://attack.mitre.org/> . Then extract words from ranked phrases of detections and filter all the bad words (words which are not necessary). Thus, keywords are extracted. Now searching for anchor tags in tables helps to find mitigations of respective links.

The following functions are used in extraction of detections, keywords and mitigations.

* def remove\_html\_tags(text)

This function removes all the html tags using re library. As all the html tags are between ‘<’ and ‘>’ symbols, the content between those symbols are removed.

def remove\_html\_tags(text):

clean = re.compile('<.\*?>')

return re.sub(clean, '', text)

* def remove(string)

This function removes all the white spaces from a string. It replaces all the white places by non-character.

def remove(string):

return string.replace(" ", "")

* def file\_to\_set1(file\_name)

This function filters all the links which contain 'https://attack.mitre.org/techniques/T' as a substring, thus extracting all the techniques links. It takes a file as input and returns a set.

def file\_to\_setl(file\_name):

results = set()

with open(file\_name, 'rt') as f:

for line in f:

if line.\_\_contains\_\_ ('https://attack.mitre.org/techniques/T'):

results.add(line.replace('\n', ''))

return sorted(results)

* def file\_to\_set (file\_name)

This function takes a file as input and converts all its contents to a set. The function removes all the white spaces to avoid duplicates.

def file\_to\_set(file\_name):

with open(file\_name, 'rt') as f:

st = f.read()

repl = st.replace(" ", "")

li = repl.split(',')

results = set(li)

return results

* def detection (link)

This function takes a link as input and encodes it and saves the data to url. The next line uses beautifulsoup returns the html content of a link. Then it extracts the h2 tags with id as “detection” or “detectable” from html content. The next div tag contains the detection required. This is done by text.next\_elements part of the code. This function returns the detection and also a soup (which contains the html content of the link).

def detection(link):

try:

url = urllib.request.urlopen(link).read()

soup = BeautifulSoup(url,'html.parser')

text = soup.find('h2',id = 'detection')

if text is None:

text = soup.find('h2',id = 'detectable')

if text is not None:

det = text.next\_elements

eng = str(list(det)[2])

para = remove\_html\_tags(eng)

return para,soup

else:

return None,soup

else:

det = text.next\_elements

eng = str(list(det)[2])

para = remove\_html\_tags(eng)

return para,soup

except Exception:

return None,soup

* def keywords (text, bad\_words)

This function takes a text and a list of bad words as input and returns a list of key words. It first uses a library *Rake* to extract all the words from the ranked phrases of the text. Then it filters all the bad words (the words in the list) and returns a list of keywords.

def keywords(text,bad\_words):

words = set()

r = Rake()

r.extract\_keywords\_from\_text(text)

key = dict(r.get\_word\_degrees())

for word in key:

if word not in bad\_words:

if word.isalpha() and len(word) > 3:

words.add(word)

return list(words)

* def mitigations(soup)

This function takes soup data type which contains html content. Then it extracts all the anchor tags which are inside a table (as all the mitigations are in table) and have href attributes. The names of all the related mitigations are obtained from these tags and returned as a list.

def mitigations(soup):

a\_tags = soup.select("td > a[href\*=mitigations]")

if a\_tags is not None:

miti = list()

for lin in list(a\_tags):

miti += list(lin)

return miti

else:

return None

Now moving to the application of these functions for extraction of required data.

wb = Workbook('data.xlsx')

sheet = wb.add\_worksheet()

bad\_words = list(file\_to\_set('bad\_words.txt'))

techniques = list(file\_to\_setl('attack\_links.txt'))

row = 1

for link in techniques:

print("Working on ",link)

id = link[-5:]

detcn,soup = detection(link)

mitigation = mitigations(soup)

sheet.write(row, 1, id)

sheet.write(row, 2, link)

if detcn is not None:

k = keywords(detcn,bad\_words)

print(k)

sheet.write(row, 3, detcn)

sheet.write(row, 4,list\_to\_str(k))

sheet.write(row, 5, list\_to\_str(mitigation))

row += 1

wb.close()

In this code list of bad words is taken from the file “bad\_words.txt” and the links of techniques are taken form the file “attack\_links.txt”. Then the detections are obtained by detections function. The detections text is passed as argument to keywords function for keywords and soup is passed to mitigations function for mitigations. All the acquired data is saved in an excel sheet named “data.xlsx”.

# Creating a webpage for displaying the acquired data

This part is done by django framework. Open a folder in command prompt where you want to create the django project and enter the following command.

*django-admin startproject detections*

The last part detections can be replaced by anything. It is the name of the django project. It is considered as detections in this document. Now we have to start an app which can be done by the following command. This command should be entered after navigating inside the detections folder.

*python manage.py startapp link*

The last part link can be replaced by anything. It is the name of the django app. It is considered as link in this document. Now open settings.py file in detections folder and add *‘link.apps.LinkConfig’* and *'import\_export’ in* the INSTALLED\_APPS section. Separate then by commas. Create a folder “templates” in the detections folder. All the template files are added in this folder. Now change the DIR in TEMPLATES section of settings.py file to “[os.path.join(BASE\_DIR,'templates')]”.

Now create a file urls.py in the link folder and add *path('',include('link.urls'))* in the urlpatterns section of url.py file in detections folder.

Now create a Link model in models.py file of link folder with name, link, detection, keywords and mitigate as variables. A sample has been given below. You can refer to documentation for more modifications (<https://docs.djangoproject.com/en/3.0/topics/db/models/> ).

from django.db import models

import re

class Link(models.Model):

name = models.CharField(max\_length = 20)

link = models.CharField(max\_length=200)

detection = models.TextField(null=True)

keywords = models.TextField(null=True)

mitigate = models.TextField(null=True)

def \_\_str\_\_(self):

return self.name

def get\_words(self):

res = re.findall(r'\w+', str(self.keywords))

return res

In the admin.py file import the Link model. This helps to modify the database related to Link from admin panel. In the following code the *LinkAdmin* class in inherited from *ImportExportModelAdmin* so that we import database from files directly.

from django.contrib import admin

from .models import Link

from import\_export.admin import ImportExportModelAdmin

# Register your models here.

@admin.register(Link)

class LinkAdmin(ImportExportModelAdmin):

pass

In the command prompt enter the following commands which makes changes (like adding a Link section in database) to the database.

*python manage.py makemigrations*

*python manage,py migrate*

Now create a django template using bootstrap and html which helps to display the data on the website. Create a view in views.py file and add a url path in urlpatterns section of url.py file in link folder. For more details refer to Django documentation. The links are given below.

<https://docs.djangoproject.com/en/3.0/topics/templates/>

<https://docs.djangoproject.com/en/3.0/topics/http/views/>

<https://docs.djangoproject.com/en/3.0/ref/urls/>

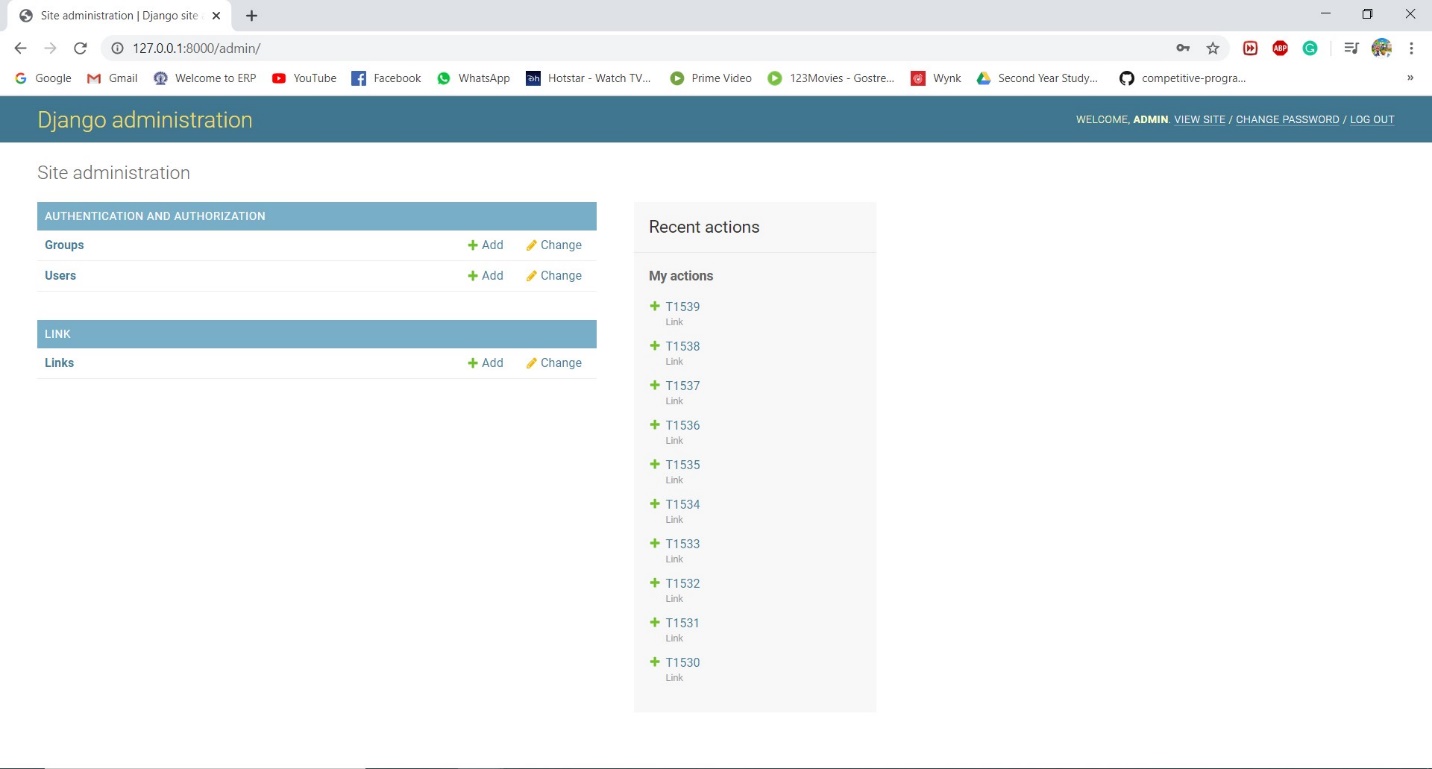
Now add create a super user. Super user helps to add the links and necessary data to the website through admin panel. Open the detections folder in command prompt. Enter the command given below.

*python manage.py createsuperuser*

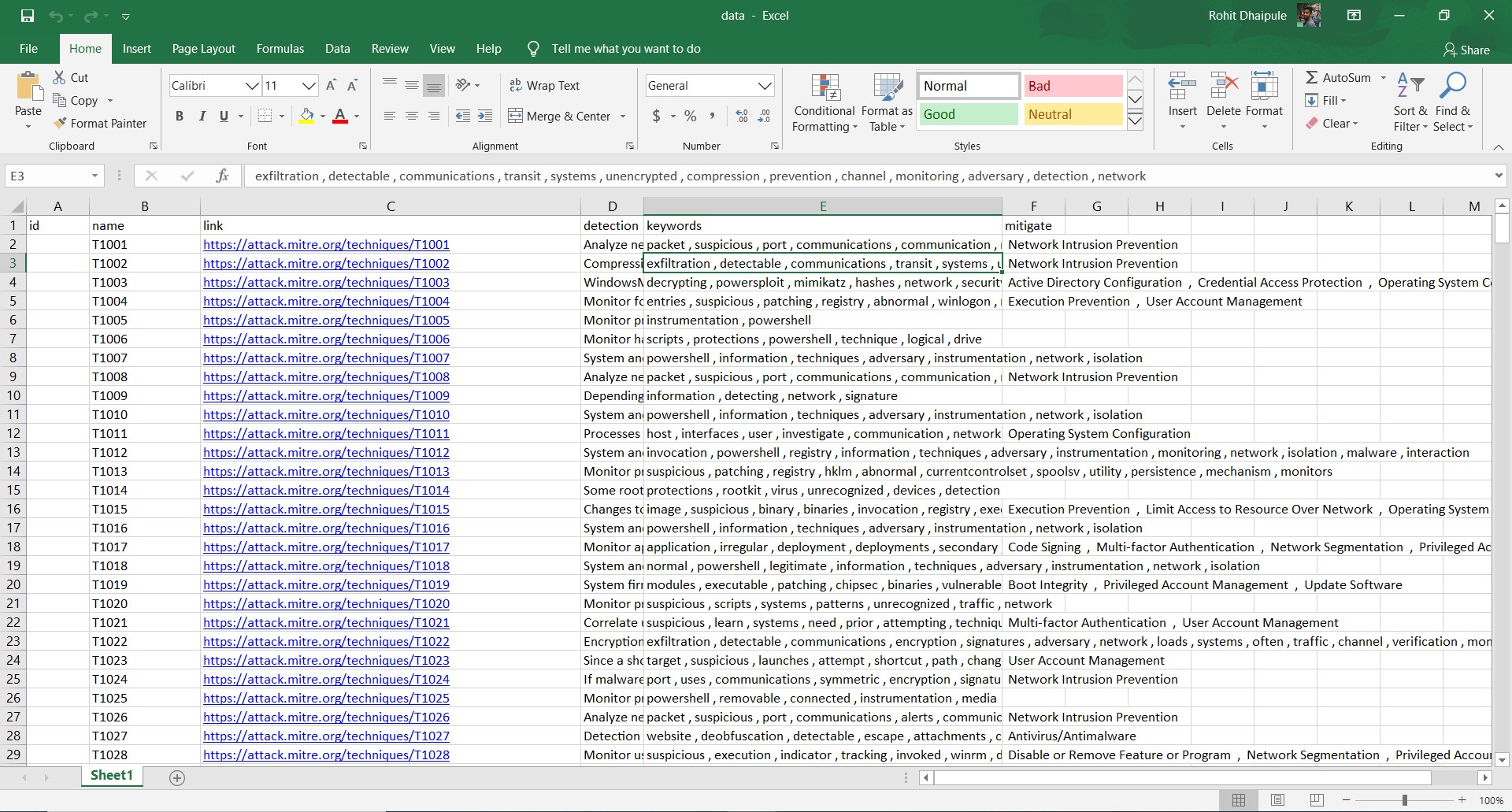
Then it asks for username, email and password. Email is an optional option but the other two are required. After entering a super user is created. Now run the local server by the following command.

*python manage.py runserver*

Then open <http://127.0.0.1:8000/admin> in any web browser. Login with your credentials and a window appears as shown below.



Now open any excel file which contains all the related data and add id, name, link, detection, keywords and mitigate in the first row without spaces as shown below.



Now click on “Links” in the admin panel and all the existing links appear. Click on import option. Upload the “data.xlsx” file and select the file type as xlsx. After clicking on submit option a preview appears. If you are satisfied then click on “Confirm Import”. All the links will be added to the database and they will be displayed on the website.

Now run the following command in the command prompt which runs a local server.

*python manage.py runserver*

Now open <http://127.0.0.1:8000/> in any web browser. The webpage with all the technique links, their mitigations, detections and their keywords are shown.