**CSCI 5742-Cybersecurity Programming-Lab 06-Web Scraping**

Name: **Lucas Fulmer**

Goals:

* Understand use of regular expressions
* Understand mechanics of web scraping and information retrieval

Deliverables

scrape.py, lab writeup – answer all questions!

One of the common tasks of a cybersecurity analyst is to parse information from either logs or the web. Sometimes there is information at sites, designed to be pulled down and parsed (NVD Database, has XML files). Other times the questions are more general like "what are the cyber blogs saying about a new vulnerability". While individuals can search a set of sites everyday, they are limited by the quantity of information that a single person can digest.

**Part I - Parsing the Web with Regular Expressions**

* Open Kali linux in NAT (If you have changed a lot of settings, you can go to the Initial Config snapshot). I would suggest changing to 1 processor with 1 core.
* Create a python file called scrape.py. Build it for Python 3.x. Here is the "main" function. Fully comment the file and build it a little at a time, uncommenting one portion at a time. Remember, you are expected to comment your code as you go along. Trust me, about a year from now, when having to make these scripts, you will hope that you had strong documentation. Again, you will learn a lot more if you type yourself rather than cutting and pasting.

**##############################  
if \_\_name\_\_ == '\_\_main\_\_':  
   
 printToFile('https://automatetheboringstuff.com/files/rj.txt', 'RomeoAndJuliet.txt')  
 printToFile('https://en.wikipedia.org/wiki/Computer\_security', 'wiki.html')  
  
 parseHTML('RomeoAndJuliet.txt', 'Juliet', 4,4)  
 parseHTML('wiki.html', 'security', 4,4)  
 justText('wiki.html')  
 justLinks('wiki.html')  
 formattedHTML('wiki.html', 'wikipretty.html')  
   
 postToForm()  
else:  
 print("imported rather than run directly")**

* First, we want to see if we can get to a website. You will need to import sys and webbrowser

**def mapIt():  
  
 if len(sys.argv) > 1:  
 #get address from command line  
 address = ' '.join(sys.argv[1])  
 else:  
 address = 'Lawrence Street Center, Denver, CO'  
  
 webbrowser.open('https://www.google.com/maps/place/' + address)**

* Now, you will need to import requests. You will need to   
   **pip3 install requests** from the command line
* Now let's go to a url and print the contents to a file

**##############################  
def webGet(url):  
  
 webpage = requests.get(url)  
 try:  
 webpage.raise\_for\_status()  
 except Exception as e:  
 print('There was a problem with the url {}'.format(e))  
 return webpage  
  
##############################  
def printToFile(url, filename):  
 webpage = webGet(url)  
 webFile = open(filename, 'wb')  
  
 for chunk in webpage.iter\_content(10000):  
 webFile.write(chunk)**

* Like we did in a previous lab with XML, you may want to format the HTML in a more human readable format
* So, let's get another library that is particularly good at parsing HTML. **pip3 install bs4**  (Beautiful Soup) (and of course **import bs4**

**##############################  
def formattedHTML(filename, formattedFilename):  
 file = open(filename, 'r+')  
 data = mmap.mmap(file.fileno(), 0)  
 html = bs4.BeautifulSoup(data, 'html.parser')  
 formatted = html.prettify('utf-8')  
 formattedFile=open(formattedFilename, "wb")  
 formattedFile.write(formatted)  
 formattedFile.close()**

* Why do we put the contents of the file into a memory map (mmap) before formatting it? **Because the information may not be ordered the way that we want it. Putting it into a memory map allows us to quickly access the fields in order to format them the way that we want.**
* Regular expressions can be valuable in parsing semi-formatted logs and unformatted, free text. It is often difficult to come up with the right regular expression, deciding between too much or too restricted information

**##############################  
def parseHTML(filename, searchWord, numWordsBefore, numWordsAfter):  
  
 print ('\*\*\*Searching in {} for the keyword {}\*\*\*'.format(filename, searchWord))  
 file = open(filename, 'r+')  
 data = mmap.mmap(file.fileno(), 0)  
  
 regexString = "(\\S+\\s+)"  
 regexString = regexString + "{" + str(numWordsBefore) + "}"  
 regexString = regexString + "\\b" + searchWord + "\\b" + "(\\S+\\s+)"  
 regexString = regexString + "{" + str(numWordsAfter) + "}"  
  
 for match in re.finditer(regexString, data.read().decode('utf-8')):  
 print('Start:{}, End:{}\n\n{}'.format(match.start(), match.end(), match.group()) )  
 file.close()**

* What is this function doing (in plain English)? **It is taking a keyword and searching the html file we have created for instances of that keyword**.
* Break down each component of the regular expression and describe each component’s purpose. **(**[**\\S+\\s**](file:///\\S+\\s)**+) is accounting for non-whitespace characters until it reaches whitespace. { + str(numwordsbefore + } captures the number of words which are contained in the file you are using.** [**\\b**](file:///\\b) **+ searchword +** [**\\b**](file:///\\b) **+** [**\\S**](file:///\\S) **+** [**\\s**](file:///\\s) **concatenates the search keyword and establishes ‘word boundaries’ around it. { str(numwordsafter) } lists the number of words following your search keyword.**
* Why do we use a \\ inside the regular expression string? **The normal command is a single \ but with python we have to use an additional backslash to account for string literals.**
* While this provided some information, many of the symbols before and after the search word were html symbols, when what we wanted was the text. (which we could combine with our previous function to parse the data.

**##############################  
def justText(filename):  
 file = open(filename, 'r+')  
 data = mmap.mmap(file.fileno(), 0) #Memory map handles the memory for large files, acts like a string  
 html = bs4.BeautifulSoup(data, 'html.parser')  
 text = html.get\_text()  
 print(text)**

* And what if we just want links to other sites.

**##############################  
def justLinks(filename):  
 file = open(filename, 'r+')  
 data = mmap.mmap(file.fileno(), 0)  
 html = bs4.BeautifulSoup(data, 'html.parser')  
 links = html.findAll("a", href=True)  
 for link in links:  
 if link['href'].startswith('http'):  
 print (link)**

**Part II - Get and Post Requests**

Accessing web sites and parsing the data is a key component of information gathering, but sometimes in order to get a list of pages etc, you have to fill out forms. Most commonly, we use a browser. That browser will send GET or POST to the server. The server will then be set up to accept or deny such requests.

* Define in your own words the difference between an HTTP GET and POST

**GET is a method to access a the information. POST is a method for assigning data to fields**

* Hopefully your Kali linux has php already loaded (if not, you would have to look up how to install php7)
* From a terminal
* **su**
* **cd /var/www/html**
* **php -ver**
* What version of php are you using? Is the latest version of php available?
* **nano test.php**

**<?php**

**phpinfo();**

**?>**

* Now go to the browser in Kali and go to <http://localhost/test.php>
* Why should you ensure that this file is deleted immediately after viewing (Hint: This is a cybersecurity course)? **It displays virtually all local network data for your machine, including ports**.
* What accounts can read, write and execute the index.html file? **Only root write. All accounts can read and execute.**
* **service apache2 start**
* **ps aux | grep apache2**
* What account owns the apache2 process? (note: the process that called the ps aux from root is always output in addition to other processes) **root**
* Why do we not just use the root account to start a web server? **Admin information will be attached to the local host**
* **nano csci5742.php**

**<?php**

**if( $\_REQUEST["name"] || $\_REQUEST["movie"] ) {**

**echo "Welcome ". $\_REQUEST['name']. "<br />";**

**echo "Your favorite movie is ". $\_REQUEST['movie']. " . <br />";**

**echo "Your Browser is ". $\_SERVER['HTTP\_USER\_AGENT'];**

**exit();**

**}**

**?>**

**<html>**

**<body>**

**<form action = "<?php $\_PHP\_SELF ?>" method = "POST">**

**Name: <input type = "text" name = "name" />**

**Age: <input type = "text" name = "movie" />**

**<input type = "submit" />**

**</form>**

**</body>**

**</html>**

* go to the Kali browser and go to http://localhost/csci5742.php
* Right click inside the name field and the inspect the element. Note the name for the name field and the name for the Favorite Movie
* Note that the method is a POST.
* Let's try a GET request (You will see from the php code below that we use php's REQUEST which allows both get and post)…use your name and your favorite movie
* [http://localhost/csci5742.php?name=’YourName’&movie=’YourFavoriteMovie](http://localhost/csci5742.php?name='YourName'&movie='YourFavoriteMovie)’
* You will see this has the same affect as hitting the submit query button. (But the button actually causes a POST instead.
* Copy the browser string after 'Your browser is' (you will need to paste it into your code soon)
* Now let's try another Get request

**https://google.com**

* Right click on the search field. Inspect the element. Note the name of the field (not the ID).
* Make a query in your browser for cybersecurity using the Get method above. https://google.com?q=cybersecurity
* You will see that this just fills in the field, but does not actually submit it (implying that it blocks Get methods). Obviously google and most search engines have figured out how to stop the amateurs from automated data calls. Google provides an API which will allow searches….some for free with advertising , some for bulk queries.
* <https://developers.google.com/custom-search/json-api/v1/overview>
* Actually, this can be a pretty cheap service for your small businesses. My recommendation is pay the fee, versus trying to find usually unsuccessful work arounds. If your organization is on the "up and up" you won't mind google tracking what you search for
* <https://enterprise.google.com/search/products/gss.html#pricing_content>
* OK, so you get the idea of GET and POST (and the potential limitations). Now we want to code using GET and POST methods

**def postToForm():  
 header = {'User-Agent': 'Paste browser from above'}  
 r = requests.post('http://localhost/csci5742.php', data={'name':'YourName','movie':'YourMovie'}, headers=header )  
  
 print (r.text)  
 print (r.headers)**

* Run it first without the headers=header (just delete the *, headers=header*. What is the default when running from python? **Python-requests/2.18.4**
* Then run it with the headers=header. Note that most decent sites will check for a current browser as the User-Agent, so at a minimum, you have to send GET and POSTs looking like a browser, not a python script.