Project 5:

Web Crawler + Form bruteforcer

Michael Rizzo, David Chen, Anuj Prajapati

109680161 | 110586427 | 110469971

## 

How We Did It:

Our project was done entirely in python. We decided to use python for its vast array of tools at its disposal and for the lack of typed variables to simplify our code and add to its readability.

At first we held a precursor meeting to decide where we stand in regards to this project and to divide the work based on our individual skill sets. At this meeting we did some research for various parts of the project and compartmentalized the project into different sections to which we could easily tie in when we finished.

Splitting up the project amongst my team members proved to be quite successful as we were quickly able to become masters of our respective parts, to which we could each fully understand if something went wrong, where it went wrong and why. We split the project up into several main parts as seen in the list below:

1. Crawling

* 1. Type of Crawling
     1. Depth-first crawl
     2. Breadth-first crawl
  2. Crawl Robots.txt and all paths not crawled
  3. Use [this](https://github.com/rbsec/dnscan/blob/master/subdomains-100.txt) to crawl more subdomains
  4. Stay on current domain
  5. Recover from dead ends & errors
  6. Identify log in page

1. Make python twisted Get/Post Wrapper class
2. Tokenize the website
   1. Form iteration utilizing beautifulsoup
   2. Prune results
   3. Word conversion
      1. Lowercase
      2. Uppercase
      3. Reverse
      4. L337
3. Form Bruteforcer

How We Met The Requirements

1. **Crawling**

Our program starts by asking the user to input the URL, crawling method (algorithm), any max depth/crawl limits and a custom user agent for user to specify for the http requests that get sent. The user inputs the URL of the starting site they wish to crawl, then they chose an algorithm for the crawl, either breadth first search (bfs) or Depth first search (dfs). If bfs is chosen the user is prompted to specify a max CRAWL limit, if dfs was chosen the user is prompted to specify a max DEPTH limit. Once those values are set a few things are done before actual running the crawl. First we parse the url to get the domain to later compare against any scraped URLs so we stay on the main domain. Then the popular subdomains from the subdomains.txt file are put into a list, appended to the domain to create a full URL. A get request is sent to the URL to see if the page exists if not the error is handled. Any valid links gets appended to the data structure the specified algorithm uses to traverse through the sites. The same thing is done with robots.txt. It is crawled and takes any sites not in the data structure and puts it in there.

Now depending on the specified algorithm either the dfs\_crawler function will be called or bfs\_crawler function is called. The bfs and dfs algorithm are based off of a common bfs/dfs algorithms online except with the base case changed to having the recursive loop stop when the MAX crawl or depth limit is reached. The algorithms scrape a root site and find all the hyperlinks in the site using Beautifulsoup to parse the raw HTML body. Any links not on the main domain are eliminated. Check if the page has been visited if not add the url to the data structure and take out from the queue or pop depending on the algorithm, then just recursively call that function on the next url in the traversal (dependent of the algorithm). All of the sites that were crawled are dumped into a list of URLs which are then sent to the tokenizer to scrape the words.

2. **Network Socket HTTP requests**

We utilized Python 3’s native socket library (import socket) to facilitate our requests. This low-level networking interface was used to send a request (GET or POST) to a given URL on port 80. For our use case, we did not feel it was necessary to recreate all HTTP method requests possible therefore we created what we needed, namely POST with a method body (and without) and GET a given url. We send a request out with the intention of reading in a website <250000 bytes. If a post request is sent out the HTTP header is encoded in utf8 and the body is encoded in ascii. If a get request is sent out the header is encoded in utf8.

* + Sends raw HTTP requests using a string-building algorithm
  + Has the means to create raw HTTP POST and GET requests, encode them to send them over the network and decode them to read them once we have received them over the network.
  + Handles network errors such as 4xx/5xx graciously by returning an ERROR string code to the caller of the request method.

3. **Tokenize**

Once the pages are crawled, they are stored in a master list of URLs. The tokenizer uses Beautifulsoup to extract words from a page. Then, we go through the strings and extract each individual string by splitting the extract words on spaces. Next, we prune the strings by removing all special characters through a regex operation. All empty strings are then removed(strings that contained only special characters). Finally, we convert the pruned wordlist to uppercase, lowercase, reverse, and 1337 speak and append them all to a master wordlist that will be later used as the database for the bruteforcer.

4. **Bruteforcer**

The bruteforcer requires a wordlist and the login url. The wordlist is provided through the tokenizer, and the login url is found by searching through the url pool for the string “login”. The bruteforcer attempts every combination of words as user and password from the wordlist. It detects if a login is successful by checking the current HTML page and searching for the word “Dashboard”. Since the website we are cracking is a wordpress site, a successful login will take you to the dashboard page. If a login page is not detected, the brute forcer will not run.

Notes:

The testing site we used is http://54.174.255.69/

It was hosted using AWS and made using WordPress.

Set Up Instructions(Windows only)

1. Install Python and BeautifulSoup 4.
2. Download our zip file with the bruteforcer from Google Drive : Link Here
3. To run our project place the downloaded file in a directory that can execute python commands.
4. Unzip the file.
5. Ensure you have the following files in one directory:
   1. webcrawlerbruteforcer.py
   2. subdomains.txt
6. Run a python-enabled console and navigate to the directory of the unzipped folder.
7. Execute the script by running the command:

py webcrawlerbruteforcer.py

Note: If the brute forcer stops randomly try utilizing a vpn or changing to a non-public network or another network. It is likely you are being rate-limited by some higher entity.

Contributions:

Anuj Prajapati: Breadth First Crawler, Depth First Crawler Internals, Report

David Chen: Depth First Crawler, Tokenizer, Brute Forcer, Report

Michael Rizzo: Network Socket Request I/O, Report

References:

https://www.dataquest.io/blog/web-scraping-tutorial-python/

https://stackoverflow.com/questions/33367439/scrapying-a-form-using-beautifulsoup

https://stackoverflow.com/questions/44441518/getting-all-the-visible-text-from-a-webpage-with-beautiful-soup-and-python

https://en.wikipedia.org/wiki/Depth-first\_search

http://www.cis.uni-muenchen.de/~yeong/Kurse/ss09/WebDataMining/kap8\_rev.pdf

https://github.com/n00py/WPForce