CS5700 Project

Web Crawler

Description

This assignment is intended to familiarize you with the HTTP protocol. HTTP is (arguably) the most important application level protocol on the Internet today: the Web runs on HTTP, and increasingly other applications use HTTP as well (including Bittorrent, streaming video, Facebook and Twitter's social APIs, etc.). Your goal in this assignment is to implement a web crawler that gathers data from a fake social networking website that we have set up for you. The site is available at: Fakebook.

What is a Web Crawler?

A web crawler (sometimes known as a robot, a spider, or a screen scraper) is a piece of software that automatically gathers and traverses documents on the web. For example, lets say you have a crawler and you tell it to start at www.wikipedia.com. The software will first download the Wikipedia homepage, then it will parse the HTML and locate all hyperlinks (i.e. anchor tags) embedded in the page. The crawler then downloads all the HTML pages specified by the URLs on the homepage, and parses them looking for more hyperlinks. This process continues until all of the pages on Wikipedia are downloaded and parsed.

Web crawlers are a fundamental component of today's web. For example, Googlebot is Google's web crawler. Googlebot is constantly scouring the web, downloading pages in search of new and updated content. All of this data forms the backbone of Google's search engine infrastructure.

Fakebook

We have set up a fake social network for this project called Fakebook. Fakebook is a very simple website that consists of the following pages:

- **Homepage**: The Fakebook homepage displays some welcome text, as well as links to several random Fakebook users' personal profiles.
- **Personal Profiles**: Each Fakebook user has a profile page that includes their name, some basic demographic information, as well as a link to their list of friends.
- **Friends List**: Each Fakebook user is friends with one or more other Fakebook users. This page lists the user's friends and has links to their personal profiles.

In order to browse Fakebook, you must first login with a username and password. Each student in CS5700 has an account created in Fakebook; your **username** is your **NU username** and your **password is your NUID (with any leading zeros)**. For example, if your NU email was "l.hamandi@northeastern.edu" and your NUID was 001234567, your Fakebook username would be "l.hamandi" and your password would be "001234567".

WARNING: DO NOT TEST YOUR CRAWLERS ON PUBLIC WEBSITES

Many web server administrators view crawlers as a nuisance, and they get very mad if they see strange crawlers traversing their sites. **Only test your crawler against Fakebook, do not test it against any other websites**.

High Level Requirements

Your goal is to collect ***5 secret flags*** that have been hidden somewhere on the Fakebook website. The flags are unique for each student, and the pages that contain the flags will be different for each student. Since you have no idea what pages the secret flags will appear on, your only option is to write a web crawler that will traverse Fakebook and locate your flags.

Your web crawler **must** execute on the command line using the following syntax:

```
./webcrawler [username] [password]
```

username and *password* are used by your crawler to log-in to Fakebook. You may assume that the root page for Fakebook is available at https://project2.5700.network/ . You may also assume that the log-in form for Fakebook is available at https://project2.5700.network/accounts/login/?next=/fakebook/ .

Your web crawler should print, to the screen, **exactly fives lines of output**: the five *secret flags* discovered during the crawl of Fakebook. If your program encounters an unrecoverable error, it may print an error message before terminating.

Secret flags may be hidden on any page on Fakebook, and their relative location on each page may be different. Each secret flag is a 64 character long sequence of random alphanumerics. All secret flags will appear in the following format (which makes them easy to identify):

```
<h2 class='secret_flag' style="color:red">FLAG: 64-characters-of-random-alphanumerics</h2>
```

The program should look for that specific HTML format when searching for the secret flags. You need to parse the HTML for finding these flags.

There are a few key things that all web crawlers must do in order to function:

- **Track the Frontier**: As your crawler traverses Fakebook it will observe many URLs. Typically, these uncrawled URLs are stored in a queue, stack, or list until the crawler is ready to visit them. These uncrawled URLs are known as the frontier.
- **Watch Out for Loops**: Your crawler needs to keep track of where it has been, i.e. the URLs that it has already crawled. Obviously, it isn't efficient to revisit the same pages over and over again. If your crawler does not keep track of where it has been, it will almost certainly enter an infinite loop. For example, if users A and B are friends on Fakebook, then that means A's page links to B, and B's page links to A. Unless the crawler is smart, it will ping-pong back and forth going A->B, B->A, A->B, B->A, ..., etc.
- **Only Crawl The Target Domain**: Web pages may include links that point to arbitrary domains (e.g. a link on google.com that points to cnn.com). **Your crawler must only traverse URLs that point to pages on webcrawler-site.ccs.neu.edu**. For example, it would be valid to crawl *https://project2.5700.network/fakebook/018912/*, but it would not be valid to crawl *http://www.facebook.com/018912/*.

In order to build a successful web crawler, you will need to handle several different aspects of the HTTP protocol:

- HTTP GET - These requests are necessary for downloading HTML pages.
- HTTP POST - You will need to implement HTTP POST so that your code can login to Fakebook. As shown above, you will pass a username and password to your crawler on the command line. The crawler will then use these values as parameters in an HTTP POST in order to log-in to Fakebook.
- Cookie Management - Fakebook uses cookies to track whether clients are logged in to the site. If your crawler successfully logs in to Fakebook using an HTTP POST, Fakebook will return a session cookie to your crawler. Your crawler should store this cookie, and submit it along with each HTTP GET request as it crawls Fakebook. If your crawler fails to handle cookies properly, then your software will not be able to successfully crawl Fakebook.

In addition to crawling Fakebook, your web crawler must be able to correctly handle HTTP status codes. Obviously, you need to handle 200, since that means everything is okay. Your code must also handle:

- 301 - Moved Permanently: This is known as an HTTP redirect. Your crawler should try the request again using the new URL given by the server.

- 403 - Forbidden and 404 - Not Found: In this case, your crawler should abandon the URL that generated the error code.
- 500 - Internal Server Error: Indicates that the Server could not or would not handle the request from the client. In this case, your crawler should re-try the request for the URL until the request is successful. Our web server may **randomly** return this error code to your crawler.

Logging in to Fakebook

In order to write code that can successfully log-in to Fakebook, you will need to reverse engineer the HTML form on the log-in page. Students should carefully inspect the form's code, since it may not be as simple as it initially appears.

Language

Write your code in Python. Your code should compile and run on **unmodified** Khoury Linux machines **on the command line**. Do not use libraries that are not installed by default on the Khoury Linux machines. Similarly, your code must compile and run on the command line. You may use IDEs like spyder, pycharm, etc. during development. Make sure your code has **no dependencies** on your IDE.

Legal Libraries and Modules

Students may use any available libraries to create socket connections, parse URLs, and parse HTML. However, **all HTTP request code must be written by the student, from scratch**. Your code must build all HTTP messages, parse HTTP responses, and manage all cookies.

For example, the following Python modules are all allowed: *socket*, *urllib.parse*, *html*, *html.parse*, and *xml*. However, the following modules are **not** allowed: *urllib*, *urllib2 (deprecated)*, *Requests, BeautifulSoup, http.cookiejar*, and *http.cookies*.

The listings above are not exhaustive. If students have any questions about the legality of any libraries send an email to me and to the TAs. It is much safer to ask ahead of time, rather than turn in code that uses a questionable library and receive points off for the assignment after the fact.

Submitting Your Project

We will use gradescope to handle submissions of your project code.

To turn-in your project, you should submit the following:

- Your (thoroughly documented) Python code. Your code should be well commented. Mention the uses of each function, what arguments it receives, and what output it generates. **Name your executable webcrawler**. So your code can run using: ./webcrawler username password
- A plain-text (no Word or PDF) **README.md** file. In this file, you should briefly describe your high-level approach (all steps involved in logging in, crawling and getting the secret Flags), any challenges you faced, and an overview of how you tested your code. You must also include a breakdown of who worked on what part(s) of the code. Also, give us the steps on how to run your code.
- A file called **secret_flags** (don't write anything in this file)
- A **Makefile** (even if you don't change anything in the Makefile given with the starter code)

Your README.md, secret_flags file, source code (webcrawler executable), etc. should all be placed in a directory. You submit your project as a zip of that directory to Gradescope. **While, only one group member needs to submit your project, the submitter must add the other group members to the submission.** Your group may submit as many times as you wish. Only the last submission will be graded, and the time of the last submission will determine whether your assignment is late. **You must add your teammate EVERY TIME you submit your code to Gradescope! Don't forget about that!**

Grading

This project is worth 50 points and it constitutes 10% of your final grade. You will receive full credit if:

1. Your code compiles, runs, and produces the expected output
2. You have not used any illegal libraries
3. You successfully submit the *secret flags* of all group members using the above requirements
4. The README.md is clear and addresses all the items requested
5. Your Python code is readable, well-documented, and passes other manual review checks
6. Your code is authentic and not similar to any other student's code or any online source.

All student code will be scanned by plagarism detection software to ensure that students are not copying code from the Internet or each other.

In addition, you must include all required files, with filenames and content exactly as specified above; your code must be documented using useful comments; and your code must produce correct output for any valid username/password, *not just yours*.

References

I highly recommend the HTTP Made Really Easy tutorial as a starting place for students to learn about the HTTP protocol.

HTTP Made Really Easy

python-socket-network-programming

socket-programming-python

python_networking

html.parser library

The developer tools built-in to the Chrome browser and
The Firebug extension for Firefox, are both excellent tools for inspecting and understanding HTTP requests.
Firefox's developer tools can show raw mode for HTTP requests and responses (there is a Raw toggle on each message). Here is a link to the documentation for the Network pane of Firefox's Developer tools: https://firefox-source-docs.mozilla.org/devtools-user/network_monitor/request_details/index.html