CS4700/CS5700 Network Fundamentals
Project 1: Web Crawler

Submission deadline: December 2nd, 11:59pm.

0.1 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 here: Fakebook.

0.2 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.

0.3 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. We will email each student to give them a unique username and password.

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.**

0.4 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

http://fring.ccs.neu.edu/fakebook/.

You may also assume that the log-in form for Fakebook is available at

http://fring.ccs.neu.edu/accounts/login/?next=/fakebook/.

Your web crawler should print **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 sequences 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>
```

0.5 Implementation Details and Hints

There are a few key things that all web crawlers must do in order 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 should only traverse URLs that point to pages on** fring.ccs.neu.edu. For example, it would be valid to crawl `http://fring.ccs.neu.edu/fakebook/018912/`, but it would not be valid to crawl `http://www.facebook.com/018912/`. Your code should check to make sure that each URL has a valid domain (i.e. the domain is `fring.ccs.neu.edu`) before you attempt to visit it.

In this assignment, your crawler must implement HTTP/1.1 (not 0.9 or 1.0). This means that there are certain HTTP headers like *Host* that you must include in your requests (i.e. they are required for all HTTP/1.1 requests). We encourage you to implement *Connection: Keep-Alive* (i.e. pipelining) to improve your crawlers performance (and lighten the load on our server), but this is not requiring, and it is tricky to get correct. We also encourage students to implement *Accept-Encoding: gzip* (i.e. compressed HTTP responses), since this will also improve performance for everyone, but this is also not required. If you want to get really crazy, you can definitely speed up your crawler by using multithreading or multiprocessing, but again this is not required functionality.

One of the key differences between HTTP/1.0 and HTTP/1.1 is that the latter supports *chunked encoding*. HTTP/1.1 servers may break up large responses into chunks, and it is the client's responsibility to reconstruct the data by combining the chunks. Our server may return chunked responses, which means your client must be able to reconstruct them. To aid in debugging, you might consider using HTTP/1.0 for your initial implementation; once you have a working 1.0 implementation, you can switch to 1.1 and add support for chunked responses.

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: Our web server may return these codes in order to trip up your crawler. In this case, your crawler should abandon the URL that generated the error code.
- 500 - Internal Server Error: Our web server may **randomly** return this error code to your crawler. In this case, your crawler should re-try the request for the URL until the request is successful.

I highly recommend the HTTP Made Really Easy tutorial as a starting place for students to learn about the HTTP protocol. Furthermore, the developer tools built-in to the Chrome browser, as well as the Firebug extension for Firefox, are both excellent tools for inspecting and understanding HTTP requests.

0.6 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. The key acronym you should be on the lookout for is *CSRF*.

0.7 Language

You can write your code in whatever language you choose, as long as your code compiles and runs on **unmodified CCIS Linux machines** on the command line. Do not use libraries that are not installed by default on the CCIS Linux machines, or that are disallowed for this project. You may use IDEs (e.g. Eclipse) during development, but do not turn in your IDE project without a Makefile. Make sure you code has no dependencies on your IDE.

0.8 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.

In other words, you need to implement the ability to send HTTP/1.1 messages, and parse HTTP responses. Students may use any available libraries to create socket connections, parse URLs, and parse HTML. However, you may not use any libraries/modules/etc. that implement HTTP or manage cookies for you.

For example, if you were to write your crawler in Python, the following modules would all be allowed: *socket, parseurl, html, html.parse, and xml.* However, the following modules would not be allowed: *urllib, urllib2, httplib, and cookielib.*

Similarly, if you were to write your crawler in Java, it would **not be legal** to use *java.net.CookieHandler, java.net.CookieManager, java.net.HttpCookie, java.net.HttpUrlConnection, java.net.URLConnection, URL.openConnection(), URL.openStream(), or URL.getContent().*

If students have any questions about the legality of any libraries please post them to Blackboard. 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.

0.9 Submitting Your Project

To turn-in your project, you should submit your (thoroughly documented) code along with three other files:

- A Makefile that compiles your code.
- A plain-text (no Word or PDF) README file. In this file, you should briefly describe your high-level approach, any challenges you faced, and an overview of how you tested your code.
- A file called *secret_flags*. This file should contain full names, NU IDs and the secret flags of all group members, one per line, in plain ASCII.
- Your README, Makefile, secret_flags file, source code, etc. should all be placed in a single zip file. You submit your project by uploading your zip file to Blackboard (see Project 2 folder on Blackboard).

Only one group member needs to submit your project. There is no need to register your group just make sure your README file and secret_flags includes all the group members names and NU IDs.

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.

0.10 Grading

This project is worth 10 points. You will receive full credit if;

1. your code compiles, runs, and produces the expected output,
2. you have not used any illegal libraries, and
3. you successfully submit the secret flags of all group members

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