CRAIGSLIST-RECOMMEND HOUSES/APARTMENTS

UNIVERSITY AT ALBANY, SUNY (SPRING 14)

HARDEEP YADAV , SAHIL CHAUDHRY , SAUMITRA JAIN

Abstract— Craigslist is a popular website that provides local classifieds and forums for jobs, housing, sale and services. Since the website was started, it has given diverse users an ease to stay informative while they are searching for housing in the region they are living or might be relocating from distant area. This paper will analyze the housing/apt service of Craigslist, and provide measures to the users to access the need as per their requirements.

Index Terms—Web Search, Web Crawlers, Craigslist, Classification.

I. INTRODUCTION

Craigslist was started in 1995 originally as a simple email distribution network for advertising local events. The following year it was deployed as a web service. Since then the services offered have expanded and follow a generic paradigm of advertising the sale or willing purchase of goods and services. It is worth noting that in 2004 eBay purchased a 25% stake in the company. These two services share the mechanism of connecting buyers with sellers but employ completely different policies. Craigslist still makes the majority of its revenue from posting paid job advertisements.

The Craigslist site is first divided by region; both country and city. This encourages local exchanges which increases safety from fraud. The service is further categorized into sections such as: for sale, job postings, personals, housing, services, etc. Within these categories, anyone can read and post new postings. Posting are valid for 30 days before being removed from the site. A post consists of information on the item or service to be sold or wanted as well as some form of contact information for readers to reach the poster. Since the web service was started, it has been done in with huge amount usability that too basically on surfing housing. The amount of help literature on searching housing related and posting throughout the site is evidence that there are clearly a lot of scope and promise of this socio-Network

Our objective is to present some of housing/apt search Related issues associated with Craigslist. To this aim, we will implement, 1) a crawler to harvest houses/apt and phone numbers from Craigslist, and 2) a flagger or a classifier that will help boosting a post, based on the price, apartment type.

II. POSSIBLE Inputs

A. Assets at Search

The assets at search can be separated into different categories, depending on association with search.

- *Pricing*
- *Apartment type (2bdr, 3bdr)*
- *Location and accessibility*

With pricing user can search their requirement based on the price of house. Pricing is all dependent on the user requirement whether it be a 2bhk or else, also basic utilities are inclusive of price or are extra charged.

User can input their data for search based on size of houses. What type of housing is required by the user

based on 2bk or else and also the basic utilities included into it.

Finally, location is one the major issue on which user focus, depending on the accessibility to transportation and other means of livelihood.

CIA Threats

Confidentiality - is reduced when a crawler is used to find personal data.

Integrity - is reduced if someone uses spamming to send Craigslist emails across the internet, making Craigslist less credible

Availability - is reduced if one uses an auto flagger to remove posts on Craigslist.

System Architecture

The architecture as shown in figure 1 includes three major function blocks: Crawler, Mining Module and User Interface. The crawler retrieves posts relevant to Housing/Apt and archives them. The Mining Module indexes the collected posts, analyzes them, and supports various types of searches. The User Interface integrates the archived posts, and specific search, and the generated data from the Mining Module into an interactive interface.

LANGUAGES AND TOOLS UTILIZED IN THE PROJECT

* Crawling: Since the dataset was provided, there was no need to crawl the web sites for data. However, the dataset had to be transformed into a format that would be compatible with the indexer.

*Python Image Library (PIL): The visualization for clustering and heat map is supported by PIL

* Scikit: Data mining results (K-Means clustering and DBScan clustering) are generated by using open source data mining tool scikit.

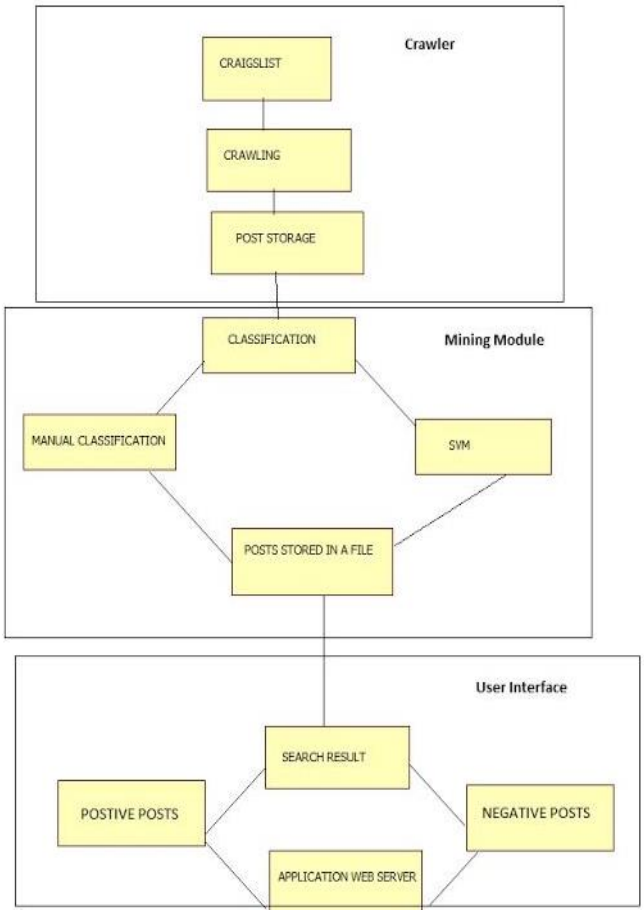


Figure 1: High Level Architecture

Separating the web scraper into pieces

There are three pieces to this application:

1. The user interface which handles input and displays out.

2. The HTTP client which actually accesses the HTML page and gets the information therein.

3. The HTML parser which reads the HTML and collects the parts we want to keep.

Python Directory Formulation

MyPost (Application Folder)

MyHttp.py (This will be responsible for the http client that gets the web pages)

MyParser.Py (To read the HTML and collects the good parts)

MyPost.Py (This will be the executable that handles user inputs and displays the results from MyParser)

Relationship Module for Dataset

Module relation describes the user input flow into the system. User logs on with personal profile to search for the requirement and skim through the system to get desired output, Mapping the dataset into positive and negative data element using classifier like SVM, using users desired Requirement.

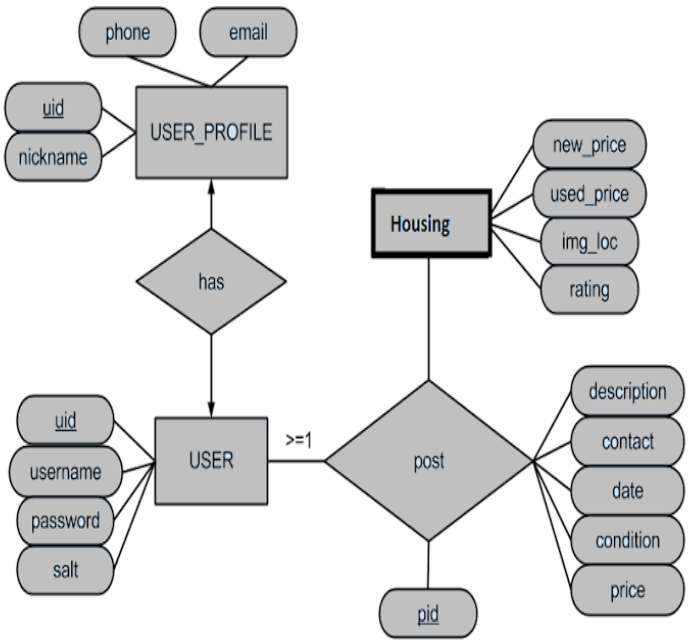


Figure 2. Relationship module

SUPPORT VECTOR MACHINES

A support vector machine constructs a hyperplane or set of hyperplanes in a high- or infinite-dimensional space, which can be used for classification, regression, or other tasks. Intuitively, a good separation is achieved by the hyperplane that has the largest distance to the nearest training data point of any class (so-called functional margin), since in general the larger the margin the lower the generalization error of the classifier.

The theory of Support Vector Machines (SVM) is a classification technique and is based on the idea of structural risk minimization (Vapnik, 1989). In many applications, SVMs have shown a great performance, rather than traditional learning machines such as neural networks and have been introduced as powerful tools for solving classification problems. A first SVM maps the entry points to a feature space of higher dimension and finds a hyperplane that separates them and maximize the margin *m* between the classes in this space. Maximizing the margin *m* is a quadratic programming problem (QP) and it can be solved by solving its dual problem by using Lagrange multipliers. Without any knowledge of the mapping, the SVM finds the optimal hyperplane using the dot product functions in the space of characteristics that are called kernels. The solution to the optimal hyperplane can be written as the combination of a few entry points are called support vectors.

For linearly separable, given a set *S* of a labeled training example $(x1,y1),\ldots(yi,xi)$,each training example$xi \in R^n$ belongs to one of two classes and has a label $yi \in \{-1, 1\}$ for $i = 1,\ldots\ldots, l$. In most cases, the

search for a suitable hyperplane in an input space is too restrictive to be of practical use. One solution to this situation is to map the input space into a feature space of higher dimension and find the optimal hyperplane there. Let $z = \Phi(x)$ the corresponding vector notation in the feature space *Z*. Being *w*, a normal vector (perpendicular to the hyperplane), we find the hyperplane $w \times z + b = 0$, defined by the pair (w,b) such that we can separate the point *xi* according to the $f(xi)= \text{sign}(w \times zi + b)$, subject to: $yi(w \times zi + b) \geq 0$.

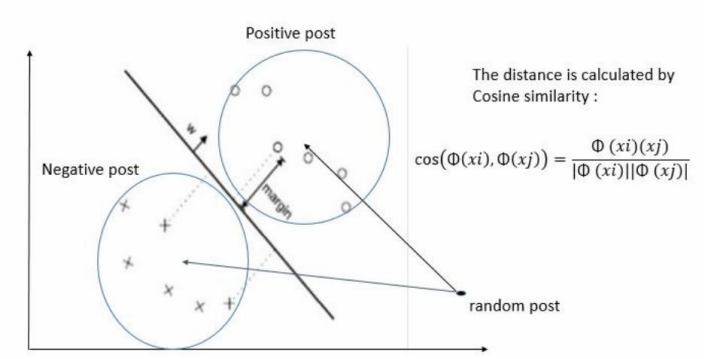


Figure 3. Separation SVM hyper-plane

In the case that the examples are not linearly separable, a variable penalty can be introduced into the objective function for mislabeled examples, obtaining an objective function $f(xi)= \text{sign}(w \times zi + b)$, subject to: $yi\ (w \times zi + b) \geq 1- \xi i$.

SVM formulations discussed so far require positive and negative examples can be separated linearly, i.e., the decision limit should be a hyperplane.

However, for many data set of real life, the decision limits are not linear.

To cope with linearly non-separable data, the same formulation and solution technique for the linear case are still in use. Just transform your data into the original space to another space (usually a much higher dimensional space) for a linear decision boundary can separate positive and negative examples in the transformed space, which is called "feature space." The original data space is called the "input space." Thus, the basic idea is that the map data in the input space *X* to a feature space *F* via a nonlinear mapping Φ,

$$X \rightarrow F$$

$$X \rightarrow \Phi(x)$$

The problem with this approach is the computational power required to transform the input data explicitly to a feature space. The number of dimensions in the feature space can be enormous. However, with some useful transformations, a reasonable number of attributes in the input space can be achieved.

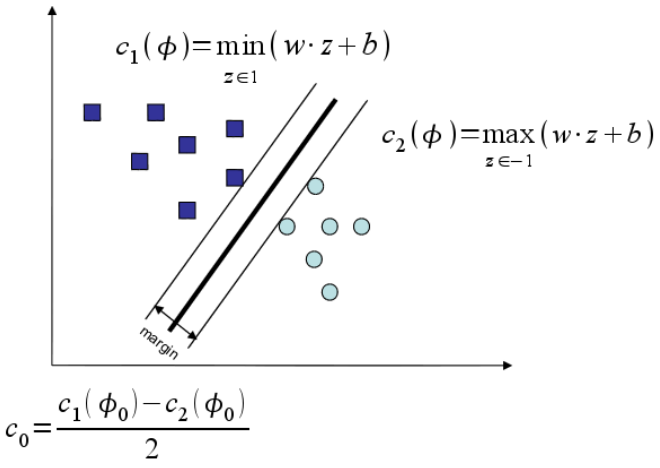


Figure 4. Separating SVM margin

Fortunately, explicit transformations can be avoided if we realize that the dual representation, both the construction of the optimal hyperplane in *F* and the corresponding function assessment decision/classification, only requires the evaluation of the scalar product $\Phi(x)\ \Phi(z)$ and the vector $\Phi(x)$ is never allocated in its explicit form. This is a crucial point. Thus, we have a way to calculate the dot product $\Phi(x)\ \Phi(z)$ in the feature space *F* using the input vectors *xyz*,

then it would not need to know the feature vector $\Phi(x)$ or even mapping function Φ. In SVM, it's done through the use of "kernel function", which is referred to as K. $K(x,z)$ equals to $\Phi(x)\ \Phi(z)$ and are exactly the functions for calculating dot products in the transformed feature space with input vectors x and z. An

example of a kernel function is the polynomial kernel, $K(x,z)=<x,z>^d$, which can replace all dot products $\Phi(x)\ \Phi(z)$. This strategy of directly using a kernel function to replace the dot products in the feature space is called "kernel trick."

Where would never have to explicitly know what function Φ is. However, the question remains how to know a kernel function without making its explicit referral. That is, ensuring that the kernel function is actually represented by the dot product of the feature space

Usage

The configuration is actually pretty simple. You should set up a DEFAULTS section with just about everything there with the exception of the search option. This will limit the amount you have to repeat yourself in your actual search configs. Note that you can override anything in the search configs and it will be used instead. For example, you could use separate dbLoc directives and use a different db file for each search, if you so choose. You could also override clBase and put in sfbay.craigslist.org if you were looking to move to San Francisco and wanted to scope out houses there.

Listed in order of importance, 1. being the most important, here is how clcheck.py will search for config files:

1. -c command-line option
2. ~/.clwatch.cfg
3. /usr/local/etc/clwatch.cfg
4. /etc/clwatch.cfg
5. ./clwatch.cfg ← config in your current directory

After the DEFAULTS section, the individual searches are defined by a name in brackets, []. The name can be anything you wish, but should probably be short since, if you are using the email option, it will be part of the subject.

Implementation

Use your browser's devtools to determine the name of the various inputs available in the search form on that page. You should end up with a list that includes at least the following:

- keywords: query=keyword+values+here
- price: minAsk=NNN maxAsk=NNN
- bedrooms: bedrooms=N

Our goal is to write a Python function that will return the search results HTML from a query to craigslist. This function should:

- It will accept one keyword argument for each of the possible query values
- It will build a dictionary of request query parameters from incoming keywords
- It will make a request to the craigslist server using this query

- It will return the body of the response if there is no error
- It will raise an error if there is a problem with the response

Next, we need a function parse_source to set up the HTML as DOM nodes for scraping. It will need to:

- Take the response body from the previous method (or some other source)
- Parse it using
- Return the parsed object for further processing.

Now, you can execute your scraper script in one of two ways:

1. python scraper.py will fetch results directly from Craigslist.
2. python scraper.py test will use your stored results from file.

Extracting Listing:

To Build function that extracts useful information from each of the listings in the parsed HTML search results. From each listing, we should extract the following information:

- Location data (latitude and longitude)
- Source link (to craiglist detailed listing)
- Description text
- Price and size data

The first job is to find the container that holds each individual listing. Use your browser's devtools to identify the container that holds each individual listing. Then, write a function that takes in the parsed HTML and returns a list of the apartment listing container nodes. Call this function extract_listings:

If you Update your __main__ block to use this new function, you can verify the results visually.

Next, Call your script from the command line (in test mode), to see your results

If you look through your test listings file using your browser's devtools, you'll notice that only *some* of them actually have latitude and longitude, Because the specs for our scraper require this data, we want to filter out any listings that do not.

```
class search_house:
    def __init__(self):

        self.promptInput()

    def promptInput(self):
        city = raw_input("Your city (all lowercase!): ");
        query = raw_input("Search your query: ");
        pricemin = raw_input("min price: ");
        pricemax = raw_input("max price: ");
        bedrooms = raw_input("bedrooms: ")
        self.search(city, query, pricemin, pricemax, bedrooms)

    def search(self, city, query, pricemin, pricemax, bedrooms):
        site = "http://" + city + ".craigslist.org/search/" + "hhh?zoomToPosting
        payload = {"query": query}
        r = requests.get(site, params=payload)
        print("URL for your query: " + r.url + "\n")

        soup = BS(r.text)

        htmlList = soup.find(id="toc_rows").findAll("p", class_="row")

        self.displaySearchResults(htmlList)
        webbrowser.open_new('C:\Users\Saumitra\Documents\codes\craigslistpos.txt
        webbrowser.open_new('C:\Users\Saumitra\Documents\codes\craigslistnef.txt
        print ""


    def displaySearchResults(self, htmlList):
        fname = 'craigslistpos.txt'
        gname = 'craigslistnef.txt'
        fname = os.path.join(out_folder,fname)
        gname = os.path.join(out_folder,gname)
        f = open(fname,'w')
        g = open(gname,'w')
        for item in htmlList:
            date = item.find("span", class_="pl").find("span", class_="date")
            if date != None:
                date = date.get_text()
                print("Date: " + date)
```

Extracting Location Data

Now that only those results that have locations are being listed, let's begin scraping that data out of the HTML page. The HTML attributes of a given tag are found as the attrs attribute of the Tag object. This attribute is a dictionary and it is certain to be present, even if it is

empty. The names of the attributes are the keys of this dictionary, and the HTML values are the values.

```
import functions
from flask import Flask, render_template, request, redirect, url_for, abort, session

app = Flask(__name__)
app.config['SECRET_KEY'] = 'F34TF$($e34D';

@app.route('/')
def home():
    return render_template('index.html')


@app.route('/search', methods=['POST'])
def signup():

    city = request.form['homecity']
    query = request.form['query']
    pricemin = request.form['pricemin']
    pricemax = request.form['pricemax']
    bedrooms = request.form['bedrooms']
    return redirect(url_for('message'))
"""
    search(self, city, query, pricemin, pricemax, bedrooms)
  """
@app.route('/message')
def message():
    render_template('message.html',city=session['homecity'],query=session['query'],pri


if __name__ == '__main__':
    app.run()
```

Extract Description Link:

We used the find_all method of a Tag above to extract *all* the listings from our parsed document. We can use the find method of each listing to find *the first* item that matches our search filter.

Use your browser's devtools to find the element in each listing that contains the descriptive text about the listing.

What kind of HTML tag is it? What other useful bit of information is present in that tag?

Use this information to enhance our extract_listings function so that each dictionary it produces also contains a description and link:Note that we are calling the string strip method on the value we get for the description from the string attribute of the Tag.The most obvious reason is that we don't want extra whitespace.The second reason is more subtle, but also more important. The values returned by string are **not** simple unicode strings.They are actually instances of the NavigableString class:These class instances contain not only the text, but also instance attributes that connect them to the DOM nodes that surround them. These attributes take quite a bit of memeory. Calling stripor casting them to unicode with the unicode type object converts them, saving memory.Executing the script from the command line now should show you that you have succeeded:

Extract Size and Prize

Again, use your browser devtools to find the container that holds *both* the price of a listed apartment, and the text that describes its size.

What's different about these two?

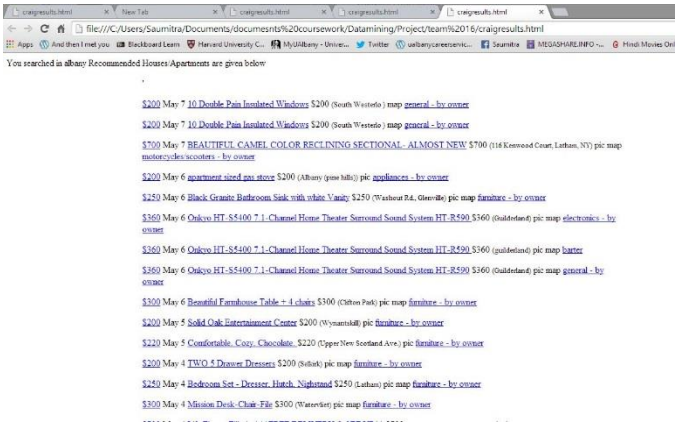
The price data is contained inside a convenient container of its own. The size, however, is not. It is just some text found in the main container **after** the Tag that holds the price. You can see this by dropping a breakpoint into your extract_listings function and inspecting the DOM: All text in a DOM document is really contained in instances of the NavigableString class.

Supports navigating from node to node in a number of ways

- **into** (or down to the next DOM tree level):
    - Tag.children (iterator with immediately contained elements)
    - Tag.descendants (generator returning **all** contained elements)
- **out** (or up to the next DOM tree level):
    - Tag.parent: (the tag containing this tag)

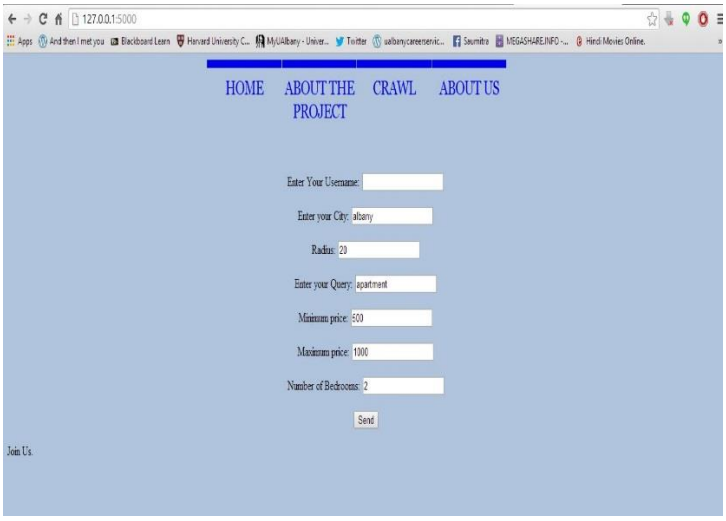
- Tag.parents: (generator returning all containers above this tag, closest first)

- **across** (or within the same DOM tree level):
  - Tag.next_sibling (the node immediately following this one)
  - Tag.next_siblings (generator returning **all nodes** at this level after this one)
  - Tag.previous_sibling (the node immediately before this one)
  - Tag.previous_siblings (generator returning **all nodes** at this level before this one)



Type 'quit' at your pdb prompt to exit the debugger and then remove the breakpoint from your code.Now update extract_listings to include the information we've just found: And now executing your script from the command line should show these new elements for a listing

Interface

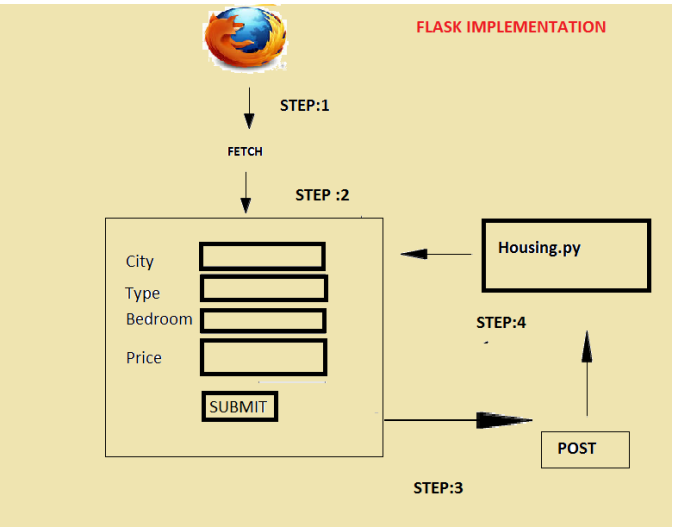


Based on Positive and Negative classification

Flask Overview:

Flask takes the flexible Python programming language and provides a simple template for web development. Once imported into Python, Flask can be used to save time building web applications. An example of an application that is powered by the Flask framework is the community page of flask.

Flask is called a micro-framework because it keeps the core simple but extensible. There is no database abstraction layer, form validation, or any other components where third-party libraries already exist to provide common functionality



FLAGGING

One of the vulnerabilities in Craigslist is lack of security around the flagging mechanism used to determine whether or not a post is useful or spam. Because the system is designed to be open to the public as well as moderated by the public, everyone has the ability to flag posts. Post can be flagged as: mis-categorized, prohibited, over post/spam or best of Craigslist. Craigslist bases the system on the assumption that you get a rough approximation of the true status of a post (good or spam) given an average over a set of votes. This assumption; however, depends on the assumption that you can trust the voters.The flagging system can be misused to do one of two things. One can flag a post as best of Craigslist even though it is not. This option can help boost a post, resulting in postings which are not necessarily helpful, residing in the best of' listings section. This removes a spot in the listings for a post which is actually helpful, and it generates undeserved advertising for the poster. This vulnerability could be exploited by an advertising company. The company could post an add and flag themselves as best of Craigslist enough times that the system puts them in the 'best of' listings. The other misuse of the system is slightly more malicious. One can flag posts as spam even though they are not. This results in posts being removed from the system which are legitimate. This vulnerability could be exploited by any user which wishes to remove competition from the system. Simply flagging any competitive posts enough times will have them removed; only leaving the user's post visible to visitors to the site. Keep in mind that advertising is one of the major sources of revenue for major companies like Google and even Craigslist itself.

The asset value of exploiting Craigslist for free advertising is higher than you might expect and the flagging mechanism lends itself to these kinds of exploits. In order to appreciate the magnitude of the vulnerability in the system, we designed a tool for use in auto flagging postson Craigslist. We chose to exploit the system in order to negatively flag posts in order to get them automatically removed. As mentioned before, the flagging mechanism does not employ any external anti spam techniques, so anyone including a machine can simply perform an HTTP post operation to the form dedicated for flagging. This form requires two inputs: the post id which is the unique numeric identifier attached to each post in the system and a flag code. Each category of flag has its own numeric code value i.e. flagging as spam has a flag code of 15 and flagging as best of Craigslist has a code of 9. Once the form is submitted to the server, the response is a webpage which contains the words "Thanks for flagging". This pattern in the response can be checked by a program in order to ensure that the flag was submitted. The final tool created was based on the Craigslist crawler previously described. The program takes as input a region and category of interest as well as a user entered string to search for. The program begins crawling the site and when it reaches a post with text matching the input string it submits a flag form using the posting id and the flag code for a spam post. The output of the application is a list of posts which were flagged as well as statistics of post flagged versus posts crawled

CONCLUSION

After the initial analysis of the Craigslist, it became clear that, although simple, the system does provide opportunities for malicious users to abuse certain features. The asset value in these cases was determined to be high enough to warrant sufficient protective measures. At this point Craigslist's position on the matter of security seems to be to provide as many warning notices and much

internet safety literature as possible in an effort to push the responsibility onto the user. This can be effective especially with a system as simple in design as Craigslist; however, it assumes that the users read and understand these warnings. As the popularity of Craigslist increases worldwide, experiments like the ones we performed for this project become increasingly pertinent. These types of experiments are important to prove not only that the vulnerabilities exist and can be exploited fairly easily, but that the users who use the system are not taking the issue of computer security particularly seriously or simply do not understand the risk. Just as companies sometimes enforce a certain level of password security used by their employees, sometimes it is best to remove some of the responsibility from the user and place it on the policies of the system itself. We believe that the countermeasures mentioned earlier are a good start towards this goal. We believe that we have done a more formal security analysis of Craigslist. Most of the existing information deals with scams and fake postings. Our crawling tool is unique such that it intelligently harvests information by location and category. In combination with our email spammer, it can be considered different than any of the existing software. We also found some useful statistics on email and phone number post rates and the reply rate from anonymous people with ambiguous email. In the process, we also found a lot of security features which Craigslist did well to prevent users from exploiting their site further.

REFERENCES

[1] M. Stamp, *Information Security Principles and Practices*. John Wiley and Sons, 2006.

[2] M. Young, *The Techincal Writers Handbook*. Ross J. Anderson, 2008.

[3] Wikipedia. Craigslist. Internet. http://en.wikipedia.org/wiki/craigslist. [Dec. 05, 2009]

[4] EECE 412, "Introduction to Computer Security," http://courses.ece.ubc.ca/412/sessions/EECE_412-02-introduction-printable.pdf , [Dec. 05, 2009]

[5] EECE 412, "Principles of Designing Secure Systems, " http://courses.ece.ubc.ca/412/sessions/EECE_412-08-design_principles-printable.pdf , [Dec. 05, 2009]