# Implementation of a Focused Social Networking Crawler

Alice Leung, Roven Lin, Jesse Ng, Philip Szeto
luciaengel@gmail.com, haskida@gmail.com, jesslikyanng@gmail.com, szeto.philip@gmail.com

***Abstract*— Social networking sites are becoming more and more popular and thus there is increased value in attacking and exploiting them. The amount of users on them is attractive in terms of the information they make available. We implement a focused social networking crawler on the popular site, Facebook, in order to exploit user profile information and identify aspects of computer security that can be improved both by Facebook and general users. We analyze HTTP packets for standard actions between a user and Facebook such as login and sending friend requests. A focused crawler is implemented in Python and used to establish a pool of friends on a fake profile. Profile information for friends is processed and statistical graphs are generated also using scripts. We identified and encountered defense mechanisms implemented by Facebook that we previously have not been aware of. Suggestions as to how Facebook can be improved and how users can prevent social engineering attacks are also presented.**

***Index Terms*—Crawler, Facebook, Focused, Security, Social-Networking**

## I. Introduction

THE following paper will outline the work that we have done and the results we saw for our term project. Our project is based on the work done by a group of students from EURECOM, an engineering school of communications, and their system iCloner [1] (identity cloner).

We implemented a system similar to iCloner in order to exploit users on a social networking site into giving us access to their profile information. The main goal of our project was to improve on the intrinsic “random” behavior of crawlers and incorporate a “focused” [2, 3, 4] mode of operation in order to target specific users or groups of users on the networking site. Doing so, we were able to identify aspects of computer security that can be improved by both Facebook and general users to prevent similar social-engineering attacks.

## II. Discussion

### *A. iCloner Architecture*

Due to the similarity in nature of our project and the iCloner project done by EURECOM students in the past, we examined and analyzed their software closely to better understand the approach they took.

Crawler, identity matcher, profile creator and message sender are the four main components of the iCloner system. In the iCloner system the crawler is responsible for the collection of data of the target user on a social networking site. The identity matcher then uses that information to compare and search against other social networking sites to determine if the same user exists. The profile creator will create a new account using the information obtained from the first networking site if the target does not have an existing account. Finally, the message sender sends requests to the target’s friends on the new social networking site as part of an identity theft attack.

We took certain aspects of the iCloner system in order to attack and exploit users on a single social networking site. With the information we were able to obtain a similar attack on a user involving cloning profiles could be further explored.

### *B. Proposed Plan*

We did not want to replicate the work already accomplished by iCloner so the main goal of our project was to take the crawling aspect of the system and incorporate intelligence in how our crawler would behave. Based on parameters given to the crawler [3] it will target users on the social networking site we chose, Facebook, and attempt to gain access to user profile information. The crawler will be logged into the networking site as a fake profile that we have created with false information and pictures. A graphic user interface (GUI) is generated to allow users to use our system intuitively and easily. The information we are able to obtain from crawling will be processed and stored in a database where we can generate charts and graphs for statistical information.

We were able to accomplish these goals and identified aspects of social engineering attacks that were effective against Facebook.

### *C. Implementation*

#### *1) HTTP Analysis*

The iCloner system uses a scripting language, Python, to automate the process of crawling and communicating with the social networking site. An HTTP library can be used in Python to do transactions such as an HTTP “GET” and “POST”. These messages are required in order for the crawler to get access to profile pages as well as perform actions such as logging in and sending requests to users.

We analyzed the HTML (HyperText Markup Language) pages inside Facebook to determine where the information we wanted is kept using a packet analyzer, WireShark. The HTTP POST messages were also analyzed to identify the parameters needed in order to perform actions. The following table shows, for example, the information required for a login POST to

Facebook.

| Parameter | Value |
|---|---|
| Charset_test | %E2%82%AC%2C%C2… |
| Version | 1.0 |
| Return_session | 0 |
| Session_key_only | 0 |
| Charset_test | %E2%82%AC%2C%C2… |
| Lsd | Random 5 character string |
| Email | User login email |
| pass | User password |

Table 1: HTTP POST Parameters for a Facebook Login

Facebook profile pages from an HTTP GET are returned in an HTML format and are normally displayed to a user graphically in a web browser such as Internet Explorer. Using a Python script with an HTTP library however, the pages are returned in a text format and need to be parsed in order to obtain the relevant information we want. We examined the HTML pages and were able to find the locations of the information we wanted inside the text.

After analyzing both the packets and HTML pages for standard interactions between a user and Facebook we were able to begin writing code for our crawler.

### 2) *Python and Jython Scripts*

We chose Python as our programming language for the reason that it has a standard HTTP library that we are able to use to interact with Facebook. Python is able support object-oriented coding structures and we made use of that to create certain classes in our scripts. Jython is a combination of the Java and Python languages and allows us to use our scripts with a GUI generated with standard Java code.

The following scripts were written for our system:

i) The crawler script implemented the main functionality of our project. Within this script we have a class for the crawler and also used a “parser” class to parse hyperlinks from HTML pages that we got from Facebook. With this script we are able to login as a user on Facebook, perform a crawl based on parameters provided by a user and send requests to add friends. There is also a function within the crawler that grabs the HTML profile page of all friends for the logged in user and parses and formats particular data that will be stored in a database.

ii) We designed a GUI shown in Figure 1 below that integrates our other scripts into a user-friendly interface. The functionalities in the GUI include logging in, crawling and generating graphs on information we obtained such as gender, age and relationship status.

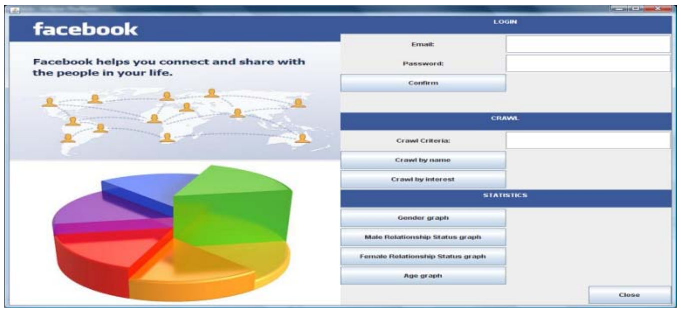

facebook
Facebook helps you connect and share with the people in your life.
LOGIN
Email:
Password:
Confirm
CRAWL
Crawl Criteria:
Crawl by name
Crawl by interest
STATISTICS
Gender graph
Male Relationship Status graph
Female Relationship Status graph
Age graph
Close

Figure 1: Graphic User Interface

A drawback we found to using Jython is that slows down the crawler script that we are running from the GUI. Preliminary investigation showed that Jython creates multiple threads when we start the GUI and at a computationally intensive part of our crawler script the processors we’re running the code on seem to stall. We compared running the script through the GUI and from a command prompt and found the stalling only happened when we ran the script from the GUI. We believe this could be an issue with Jython because the language itself is still in development.

iii) The last script in our system is a database script that is able to accept a list of information from the crawler script and input it into an SQLite databse. This script will generate the database, store the information and generate graphs based on queries.

### 3) *Algorithm*

This section will explain the algorithm for the crawler and how it differs from a standard crawler.

A typical crawler will navigate through HTML pages and gather all hyperlinks on a particular page. It will then go to each of the links found and perform the same action until it finds no links. It has an intrinsic random nature in that it does not care what sort of link it is going to. The equivalent scenario for a social networking crawler would be one that goes to all user profiles on a particular page and subsequent pages and so on.

The focused crawler we implemented is able to, with built-in functionalities in Facebook such as search, crawl intelligently through the social networking site and target profiles that a user is interested in. The crawler script we wrote takes in a parameter that we call the crawl criteria. The script will first do a search within Facebook for the input criteria and the search results form the starting point for the crawl. The HTML page from the search is parsed and the profile IDs of the users are kept and stored. Shown below is an example of what is kept from the HTML page.

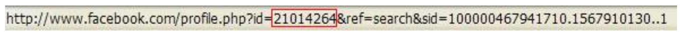

http://www.facebook.com/profile.php?id=21014264&ref=search&sid=100000467941710.1567910130..1

Figure 2: Profile ID parsed from link

If a name is chosen as the crawl parameter the crawler will search for the name and save the results in a list. It will then go to all the friends of those users and save those in a list as well. The reason for this is that we want to target a particular user's friends as well because mutual friends will make our request seem more legitimate. We found that users are more likely to accept our request if we have mutual friends.

Other crawling parameters accepted such as interests or activities return "group" or "fan" pages when using the search function in Facebook. We follow the same idea above but target the profiles within the group and fan pages for the search that we performed.

Once the crawler has a list of all the profiles we want to start we begin the phase where it adds everyone on the list. An intentional delay is added in between friend request HTTP POST messages because Facebook will identify activity such as adding friends or sending messages too quickly. Facebook will challenge the user with a CAPTCHA response in order to continue with the action. Since breaking CAPTCHA challenges was out of scope for our project we decided to simply implement a random delay of 10-20 seconds between requests. This allowed our crawler to continue adding friends without being stopped by Facebook.

As a proof-of-concept the crawler script currently only returns the first search page for the crawl and the first friends page for our targeted profiles. This is to reduce the number of profiles returned so we are able to complete running the script without having to wait a long time. Additional intelligence could be added to the crawler in the future but as part of this project we were happy with our results.

After crawling is complete we are able to process the data we have obtained for statistical information.

#### 4) *Database*

To better gather statistics on our friends, we decided to use a SQLite database to store each friend's name, gender, relationship status, and age. SQLite was chosen due to its compatibility with Jython and small manageable size. Its portability in which the resulting database can be extracted using other software or languages if so desired was also factored in our decision.

The crawling algorithm returns a list of all friends in the format of ['NAME,GENDER,RELATIONSHIP_STATUS,BIRTH_YEAR', 'NAME2,....',...], with each person as one element string in the list. For each element in the list, the database script splits the string by the comma character and inserts the four pieces of information into a PROFILE table. With the birth_year column, we simply subtract it from 2009 to obtain the age. The algorithm can be expanded in the future to insert more friend information into the table and also calculate the accurate year. Since users profile are dynamic and constantly changing, it is necessary to drop the table and insert new entries each time the database script is run. This ensures that we have the most up to date profile information.

For presentation and statistical purposes, three SQL select statements were used to extract information on Male-to-Relationship-status, Female-to-Relationship-status, and the overall age group of our friends. The pychart library, which utilizes the GhostScript software, was used to generate the three graphs. However, due to the limitation of the library, it was not possible to dynamically create a graph during runtime. The number of data entries on the axes must be determined beforehand since it cannot be appended or deleted. Another graphing library or other languages that can interact with SQLite databases should therefore be considered so the script operator does not have to manually input data points before hand. Shown below is an example of a generated graph from the database.

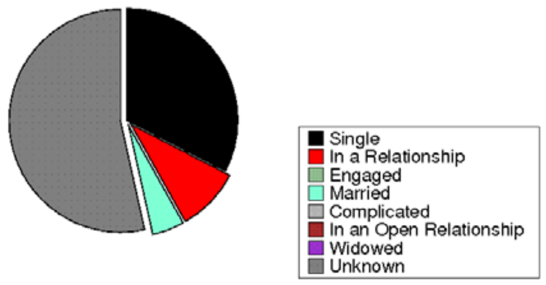


Figure 3: Generated graph on relationship status

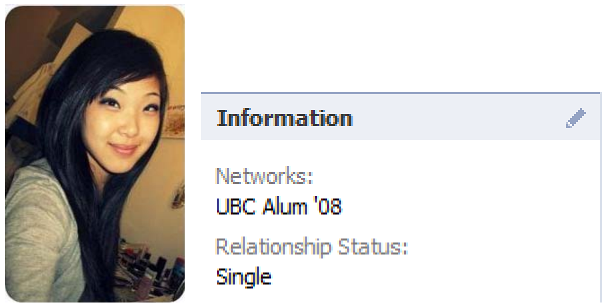
## III. Results

### A. *Results and Findings*

The following section will discuss findings that we had after completing our project. These findings can be used in the future to improve on current architecture as well as to exploit other computer security vulnerabilities within Facebook or other similar social networking sites.

#### 1) *Facebook Profile*

We created a fake Facebook account named Trudy Wong which is the center of our crawling project. Our profile picture, a young female girl with a single relationship status, is shown in Figure 4. This is the feature that attracts people into accepting our friend requests to find out more about this young lady.


Information

Networks:
UBC Alum '08

Relationship Status:
Single

Figure 4: Profile Picture and Information

Our social network attack involves targeting and adding circles of friends, and friends of friends. This created a large number of mutual friends displayed when people see our friend requests, and therefore they are less suspicious and more likely to accept our request. To date, we have sent approximately 200 friend requests, of which 130 have been accepted. This is roughly 65% acceptance rate, which is quite high considering none of these people actually know us. Out of everyone we have added, 40 people questioned our friend requests with a message “Do we know each other?” or similar. This means 160 people blindly accepted us without questioning and gave us access to their profile.

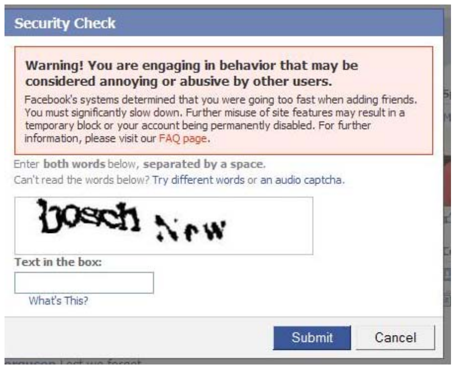
We propose a number of reasons for the high number of individuals that accepted us. As discussed above, people are willing to befriend Trudy because of the attractive profile picture. Another reason is that people do not see a bad side to adding a new friend. They do not consider clicking “accept” as losing privacy of their personal information to a complete stranger, but rather, possibly see this as a chance to boost their number of friends and social status. Of course, there are those who cannot remember if they have seen Trudy or not, but have decided to accept us out of friendliness.

We ran into a number of unique cases with our account. There were two people who actively added us, perhaps after seeing our fake profile on one of their friend’s pages. Also, there were people who started posting on our wall about our TV shows and hobbies as if we were close friends. There was also an individual who claimed he has talked to Trudy at the football game last night.

There are much more we can do with this account when we have time in the near future. Creating a pool of fake profiles and running our crawler on multiple accounts can create more links between targeted profiles. We can also attempt to “fit in” to a targeted group of users by attempting to communicate with them.

### 2) *Facebook CAPTCHA Challenge*

Facebook has implemented CAPTCHA as shown in Figure 5 when it detects a possible automated attack or suspicious behavior. We saw this when we were running the script to add a large number of friends in a relatively short amount of time. If we wait for some time and try again, this message goes away, and we can start adding friends again. In the near future, if we have time, we can explore methods of breaking this system. There are open source programs where they read the pixels to formulate the text in the CAPTCHA and break it. This can be a boost to our social network attack.



Security Check

**Warning! You are engaging in behavior that may be considered annoying or abusive by other users.**
Facebook's systems determined that you were going too fast when adding friends. You must significantly slow down. Further misuse of site features may result in a temporary block or your account being permanently disabled. For further information, please visit our FAQ page.

Enter **both words** below, **separated by a space.**
Can't read the words below? Try different words or an audio captcha.

bosch New

**Text in the box:**

What's This?

Figure 5: Facebook CAPTCHA Challenge

### 3) *Successes and Failures*

Overall, our project was quite successful. We were able to run scripts to automatically add friends, friends of friends, and therefore joined a “circle of friends” with all these mutual friends. We were then able to collect statistical information such as age and relationship status. We were able to plot them on graphs and store them into a database.

Our Facebook crawler involved both learning on the higher level analysis of social networking and the lower level details of scripting and networking. From the social aspect, we gained insight into people’s behavior and psychology of accepting friend requests. From the technical aspect, we learned how to write scripts to “crawl” through Facebook, how to parse packets sent over the internet to find friends, and how to somewhat replicate a simple version of the Facebook database using only Trudy’s friends. Our learning objectives were met.

As described above, our GUI implementation in Jython which worked against the Python script is a flaw in our project. If this is not the case, the GUI demonstration would be a huge plus to our ability to present our results. However, being able to learn about this possible bug in Jython is still a plus to our programming knowledge.

There is another minor flaw to our design in that the system needs a stable internet connection. When there is a slight connection lost, it results in a timeout and our script stops adding friends. If we have more time, we can explore ways of maintaining the state of the script even if the internet goes down for a short interval of time.

Lastly, our program currently cannot add Facebook profiles with nicknames. This is because instead of getting the profile ID when we parse through the HTML page, we see the nickname instead. Facebook added this functionality during an update they did in the past and allowed users to select a nickname. The profile ID for a user is required in order to add them as a friend so we currently don’t have anything written to get the profile ID of a user that has a nickname. More work will have to be done in order to be able to target all users. As a proof of concept we decided to leave this as a flaw with our system.

## B. *Future Expansion of Project*

As of now, our Facebook crawling project is still at an early, proof of concept stage. A major improvement to the system would be to further decrease our chance of being detected as a bot by Facebook or flagged as a spammer by potential friends. When our friend adding frequency becomes more aggressive, Facebook employs a CAPTCHA check in order to differentiate a bot and a normal human user. Currently, we only detect such a check and then delay our crawling. However, if we utilize a CAPTCHA solving method, we would be able to eliminate this delay.

Since we currently only add friends that appears in the first page of the search result as determined by Facebook. Facebook's search relevance puts people who share mutual friends, common networks/locations or those with less private profiles in the first few pages. Because of this, some people may be repeatedly added if they keep on rejecting our friend request. This can easily lead to us being flagged and exposed as a bot. To prevent this, we need to keep track of all the people we have tried to add in a separate database, then check against the table before each friend addition. Though this would slow down the process for an uncertain amount, it decreases the chance of our fake account being disabled. Additionally, we could also automate a reply to those who send private messages inquiring about our identities. A generic response such as "I know your friend Alice!" or "I like the same TV show as you!" may dissuade the potential friend from rejecting or flagging us.

The durability of our fake account is paramount as we would not be able to have access to our friends' profile if it is disabled or removed. Once the above improvements are implemented, we can also increase our efficiency in statistic gathering by modifying our database algorithm and library so that database entries insertion and graph generation can be more dynamic, as noted in the method section. We can also create another fake account of a different gender, age, or any other characteristics to compare the trend of friend acceptance to the Trudy account.

Lastly, we could implement profile cloning on networking sites where users do not currently have a profile. This method of identity theft could prove to have better results in exploiting user acceptance to friend requests because it will be sent from an actual friend of the user. The possibilities of expanding our network of crawlers could yield a vast database of information that can be used for other forms of identity attacks.

### C. *Countermeasures*

Upon crawling Facebook, we found out there are countermeasures implemented to prevent potential spam, abuse and harassment. These countermeasures are:

- There is a limit to how many people you can add per day and Facebook deliberately does not announce the number. Facebook determines the limit with factors such as speed, time, and quantity. [5] We received a warning after adding around 50 friends in one day.
- When a person receives a friend request, the person can either accept, ignore, report abuse, or mark as someone you don’t know. Presumably we’d get a warning if we receive too many flags from users. With our algorithm we were able to avoid being flagged by Facebook.
- Facebook disabled the search for friends based on their interests (i.e. clicking on your own interest such as running would bring up a list of people who enjoys running as well). Instead Facebook makes it so the user has to be a member/fan of those interests (joins the group) in order for us to get the result.

As of now, Facebook has fairly adequate protection against friend adding bots. To further protect their users from our attack, they can employ the CAPTCHA check each time a user tries to add a friend. This would significantly slow down any bots who are trying to add large amount of friends at a time, perhaps to the point which manual adding would become the only viable method. Additionally, we have already encountered a problem with Facebook changing a parameter in its POST message for adding friends. If Facebook changes the parameter for each friend request instead of weekly, this will also complicate the automation process.

### D. *Security Principle Violations*

We found that Facebook breaks the principles of: question assumptions and defence in depth. However, it does follow: least privilege, light complete mediation and psychological acceptability.

Due to the fact that Facebook is widely used and an extremely popular website, security measures are fairly well covered. There is no major violation of security principles. We’ve found five examples where we can discuss Facebook’s use of security principles.

Ease of use is important for any website and especially for a social networking site with millions of users. Facebook followed the security principle of psychological acceptability, making account management and networking very simple to use; regular users seldom come across countermeasures implemented by Facebook to prevent abuse. The ease of use sacrifices security and that leads to the two principles Facebook failed to follow: question assumption and defense in depth. A script using an HTTP library is able to interact and use Facebook as long as it exhibits human-like behavior. It is assuming then that unless specific behavior is seen then the user must be compliant with rules. Detection of script-like behavior or patterned actions could possibly be able to prevent crawlers such as ours from working with Facebook. The only defense we encountered against our crawler was the CAPTCHA challenge. We’re unsure if there are any other layers of protection after the CAPTCHAs but validation of an actual user could also help prevent crawler attacks. We see that adding extra layers of validation would eliminate the ease of use for a regular user but as Facebook stands today a crawler is able to use it fairly easily.

Though Facebook broke the two principles discussed above, it reinforces its security by following three security principles. Least privilege prevents users from modifying anything he or she is not supposed to have access to and users can only modify their own accounts or fan groups. This prevents the users from tampering around with important data or files that can affect other users. Facebook also employs a "light" complete mediation which requires inputting the user's password when changing account information.

## IV. Conclusion

Our project demonstrated a few possible achievements of social networking attacks: automatically and intelligently adding groups of friends using a bot, collecting and plotting statistical information on a graph, and creating an active database of profile information gathered from exploited users. We were successful in adding a large number of friends and through the process were able to get an understanding of how users respond to friend requests. Countermeasures and defense mechanisms were found within Facebook that changed how we implemented our crawler. With social networking sites getting more and more users we can see security begin to improve and that this topic is well-worth more exploration in the future.

From completing this project we are able to identify user habits on social networking sites and how an attack can exploit these in order to gain access to otherwise private profile information. We can also see now that there are defenses within Facebook to prevent abusive behavior but that extra work could be done to bypass those defenses.

## V. Recommendations

We recommend that social networking sites make extra effort to prevent the feasibility of social engineering attacks. We, as undergraduate students with no knowledge of crawlers when we began this project, were able to implement a system that was able to get us information on over 100 users. A more knowledgeable programmer would be able to implement a smarter crawler with more features that would be able to bypass extra levels of defense. The social network sites themselves should have a responsibility in prevent these attacks from happening.

On the other hand, for users interested in replicating our work we have the following recommendations. An understanding of CAPTCHAs is important in order to bypass the defense mechanism that we encountered. Doing so will expand the number of actions the crawler is able to perform within a particular time-frame or until the next layer of defense is activated, if any. Other programming languages could also be explored as an option to improve speed and allow the integration of scripts with GUIs.

## References

[1] Bilge, Leyla, Thorsten Strufe, Davide Balzarotti, and Engin Kirda. "All Your Contacts Are Belong to Us: Automated Identity Theft Attacks on Social Networks.". Eurecom, 20 Apr. 2009. Web. 22 Sept. 2009.

[2] B. Rashmin, “Focused Crawling,” Computer Science and Automation, pp. 1- 37, 2007

[3] O. Nasraoui, and Z. Zhang, “Profile-Based Focused Crawling for Social Media-Sharing Websites,” EURASIP Journal on Image and Video Processing, pp. 1 -13, 2009.

[4] S. Chakrabarti, B. Dom , and M. Van den berg, “Focused crawling: a new approach to topic-specific Web resource discovery,” Computer Networks 31, pp.1624 – 1640, 1999.

[5] *Facebook Warnings* Facebook, 2009. Web. 22 Sept. 2009. <http://www.facebook.com/help.php?page=421>.