Graduate School ETD Form 9
(Revised 12/07)

PURDUE UNIVERSITY
GRADUATE SCHOOL
Thesis/Dissertation Acceptance

This is to certify that the thesis/dissertation prepared

By Marvin O. Michels

Entitled
Real Time Text Analysis on Internet Relay Chat Conversations

For the degree of Master of Science

Is approved by the final examining committee:

Marcus Rogers	
Chair	
Cristina Nita-Rotaru	
Victor Raskin	

To the best of my knowledge and as understood by the student in the *Research Integrity and Copyright Disclaimer (Graduate School Form 20)*, this thesis/dissertation adheres to the provisions of Purdue University's "Policy on Integrity in Research" and the use of copyrighted material.

Approved by Major Professor(s): Marcus Rogers

Approved by: Eugene Spafford 04/26/2012
Head of the Graduate Program Date

Graduate School Form 20
(Revised 9/10)

PURDUE UNIVERSITY
GRADUATE SCHOOL

Research Integrity and Copyright Disclaimer

Title of Thesis/Dissertation:

Real Time Text Analysis on Internet Relay Chat Conversations

For the degree of Master of Science

I certify that in the preparation of this thesis, I have observed the provisions of *Purdue University Executive Memorandum No. C-22,* September 6, 1991, *Policy on Integrity in Research.**

Further, I certify that this work is free of plagiarism and all materials appearing in this thesis/dissertation have been properly quoted and attributed.

I certify that all copyrighted material incorporated into this thesis/dissertation is in compliance with the United States' copyright law and that I have received written permission from the copyright owners for my use of their work, which is beyond the scope of the law. I agree to indemnify and save harmless Purdue University from any and all claims that may be asserted or that may arise from any copyright violation.

Marvin O. Michels

Printed Name and Signature of Candidate

4/26/2012

Date (month/day/year)

*Located at http://www.purdue.edu/policies/pages/teach_res_outreach/c_22.html

REAL TIME TEXT ANALYSIS ON INTERNET RELAY CHAT CONVERSATIONS

A Thesis

Submitted to the Faculty

of

Purdue University

by

Marvin O. Michels

In Partial Fulfillment of the

Requirements for the Degree

of

Master of Science

May 2012

Purdue University

West Lafayette, Indiana

To my parents Jeff and Luetta: for raising me to overcome any obstacle in my way and giving me everything I need to succeed in life.

To the love of my life Lyssa: for pushing me through those nights where I did not want to push myself.

ACKNOWLEDGMENTS

This research would not have been possible without the support and guidance of my committee members: Dr. Marc Rogers, Prof. Victor Raskin, and Prof. Cristina Nita-Rotaru. Kyle Johansen, thank you for working with me in the Summer of 2011 or this project would have never come to pass.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Michels, Marvin O. M.S., Purdue University, May 2012. Real Time Text Analysis on Internet Relay Chat Conversations. Major Professor: Dr. Marcus K. Rogers.

Internet Relay Chat (IRC) has been and is still being used for a number of legal and illegal activities. Investigations dealing with IRC tend to be arduous and require a vast amount of man hours for the constant monitoring needed, whether it is from law enforcement or just a normal user surfing through the channels. This research looked at developing the IRC Data Gathering Tool (IRCDGT), which facilitated real-time analysis of IRC chat messages as well as real-time updates to the investigator. This is intended to help reduce the number of man-house needed in front of a computer for an investigation. A crawler was developed for IRC that goes through a list of channels and reports on what is being discussed in those channels. Normal keyword analysis statistically outperforms keyword & POST analysis in terms of recall while there is no significant difference between basic keyword analysis and keyword & POST analysis in terms of precision. Topic analysis was performed in near-real time to enhance the keyword analysis. Lastly, natural language processing seems to have issues with dealing with the language of the Internet subculture.

CHAPTER 1. INTRODUCTION

While not being the premier chat program for the Internet today, Internet Relay Chat or IRC has withstood the test of time as one of the worldwide leaders in Internet communication tools. From IRC's beginnings in the late 1980s, its use has spread worldwide and is still being used to this day to provide file sharing and communication between multiple users simultaneously. While its user base has become lessened with the onset of instant messaging and social media, many people still use IRC today for both legal and illegal purposes. Users still share stories, messages, pictures, music, software, credit card numbers, identities, child pornography, etc. If it digital data, it can be shared over IRC.

IRC is a playground for groups such as Anonymous. IRC has been used as a recruitment ground for terrorists, both physical and cyber related. The architecture of IRC provides a problem for investigators. There is no central server that all channels reside in. Rather, there are multiple servers spread around the world with thousands of channels belonging to each server. Finding those who participate in illegal activities can be a tedious task for investigators. Even if found, monitoring the room requires a large amount of man hours dedicated to monitoring, constant surveillance, and logging of all messages. Most analyses on the conversation are done post-mortem on logs of the conversation (Adams & Martell, 2008) (Dong, Hui, & He, 2006) (Elnahrawy, 2002).

This thesis looks to help in the reduction of man hours needed to facilitate an investigation on IRC and perform real-time analysis on incoming messages through a tool called the Internet Relay Chat Data Gathering Tool (IRCDGT).

1.1. Research Questions

This thesis looked at analyses that can be performed in real time on Internet Relay Chat messages. The main question here is can a tool be developed that facilitates real-time analysis of Internet Relay Chat channel discussions within acceptable time limits and with acceptable accuracy? An ancillary question to be answered, can a method be developed to find chat channels where suspicious activities are being undertaken?

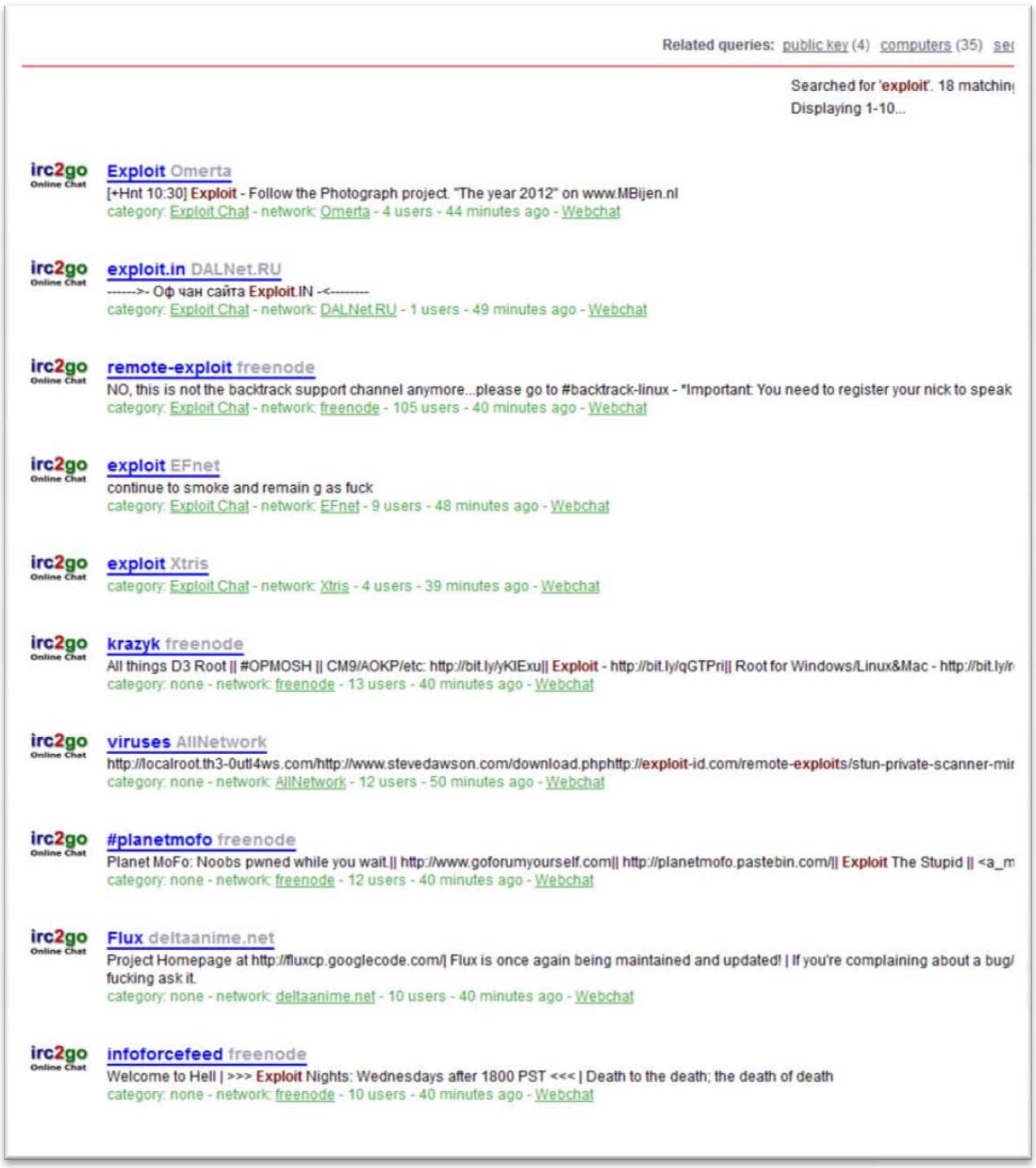
1.2. Scope

There are a number of chat networks and programs available to end-users such as AOL Instant Messenger and Skype. Each network contains a large number of users and different structures to their networks (Courtney, 2011). Internet Relay Chat will be the main target for this thesis, though the results found in this thesis can and should be applied to any chat network. The real-time analyses performed in this thesis will consist of keyword analysis, topic analysis, and keyword analysis with POST categories. The results of the real-time analyses' accuracy were broken down into precision, recall, and accuracy values.

1.3. Significance

In 2011, Internet Relay Chat had a maximum of 764,621 users spread over 4,579 servers (Gelhausen, 2012). There are still a large number of crimes and ill-practices being perpetrated on Internet Relay Chat and starting an investigation on Internet Relay Chat can be tedious (Michels, 2011). Users share exploits for different systems and hacking tools across networks, even offering tutorials on how to use said exploits. Identities, credit card numbers, and other illegal information are traded easily. Cyber-related attacks are planned and carried out through IRC (Arthur & Gallagher, 2011). Most investigations require either someone monitoring the chat room at all times or waiting till after the chatting is done and searching through large amount of chat logs and saved messages (Brown, 2007). A reduction of dedicated man hours is needed to facilitate an efficient investigation from not just a law-enforcement standpoint but from a business standpoint as well.

Law enforcement does not have the only stake in monitoring IRC chats. Many different exploits for hardware and software are shared in IRC channels. Companies can find those holes in their products quicker by keeping an eye out in specific channels. Unfortunately, due to the wide spread nature of users, channels, and servers of IRC, it is hard to find those places where the exploits are being undertaken. Popular IRC Search engines such as irc.netsplit.de or search.mibbit.com can look for phrases and keywords in channel names and return a list of channels, but might not always return valid results.


Related queries: public key (4) computers (35) se

Searched for 'exploit'. 18 matchin
Displaying 1-10...

irc2go Online Chat
Exploit Omerta
[+Hnt 10:30] Exploit - Follow the Photograph project. "The year 2012" on www.MBijen.nl
category: Exploit Chat - network: Omerta - 4 users - 44 minutes ago - Webchat

irc2go Online Chat
exploit.in DALNet.RU
------>- Оф чан сайта Exploit.IN -<--------
category: Exploit Chat - network: DALNet.RU - 1 users - 49 minutes ago - Webchat

irc2go Online Chat
remote-exploit freenode
NO, this is not the backtrack support channel anymore...please go to #backtrack-linux - *Important: You need to register your nick to speak
category: Exploit Chat - network: freenode - 105 users - 40 minutes ago - Webchat

irc2go Online Chat
exploit EFnet
continue to smoke and remain g as fuck
category: Exploit Chat - network: EFnet - 9 users - 48 minutes ago - Webchat

irc2go Online Chat
exploit Xtris
category: Exploit Chat - network: Xtris - 4 users - 39 minutes ago - Webchat

irc2go Online Chat
krazyk freenode
All things D3 Root || #OPMOSH || CM9/AOKP/etc: http://bit.ly/yKlExu|| Exploit - http://bit.ly/qGTPri|| Root for Windows/Linux&Mac - http://bit.ly/r
category: none - network: freenode - 13 users - 40 minutes ago - Webchat

irc2go Online Chat
viruses AllNetwork
http://localroot.th3-0utl4ws.com/http://www.stevedawson.com/download.phphttp://exploit-id.com/remote-exploits/stun-private-scanner-mir
category: none - network: AllNetwork - 12 users - 50 minutes ago - Webchat

irc2go Online Chat
#planetmofo freenode
Planet MoFo: Noobs pwned while you wait || http://www.goforumyourself.com|| http://planetmofo.pastebin.com/|| Exploit The Stupid || <a_m
category: none - network: freenode - 12 users - 40 minutes ago - Webchat

irc2go Online Chat
Flux deltaanime.net
Project Homepage at http://fluxcp.googlecode.com/| Flux is once again being maintained and updated! | If you're complaining about a bug/
fucking ask it.
category: none - network: deltaanime.net - 10 users - 40 minutes ago - Webchat

irc2go Online Chat
infoforcefeed freenode
Welcome to Hell | >>> Exploit Nights: Wednesdays after 1800 PST <<< | Death to the death; the death of death
category: none - network: freenode - 10 users - 40 minutes ago - Webchat

Figure 1.1 – Channel search for the word "exploit" from irc.netsplit.de

Out of the ten channels found in Figure 1.1, only two channels, krazyk freenode and viruses AllNetwork, can be expressly viewed as a channel strictly for exploits. For the other channels in Figure 1.1, it cannot be easily shown what is actually going on in the channel though they have the word exploit in their channel name or description. The channel names and descriptions cannot accurately give a description of what is going on in each channel. This is a major problem for investigators as well.

1.4. Definitions

Accuracy: The proportion of true results in a search.

Bot: An autonomously controlled client that responds to certain commands with other various actions.

Crawler : A section of code that peruses a network to create an index of data to be used at the discretion of the investigator.

False Positive: Any message containing a keyword that is incorrectly identified as suspicious will be placed as a false positive result.

False Negative: Any message containing a keyword that is not identified as suspicious when it is supposed to be will be considered a false negative

Frequency Analysis: The process of searching through a text to find the number of occurrences of each word.

Internet Chat Messaging Networks: A network that provides any type of communication over the Internet.

Internet Relay Chat (IRC): A chat network developed by Jarkko Okirinen in 1989 with capabilities for communication and file sharing.

Keyword Analysis: The process of searching through a text for the occurrence of certain keywords.

Part of Speech Tagging (POST): The process of marking up a word in a text (corpus) as corresponding to a particular part of speech, based on both its definition, as well as its context.

Peer-To-Peer Networks (P2P Network) : a computer network in which each computer in the network can act as a client or server for the other computers in the network, allowing shared access to files and peripherals without the need for a central server.

Precision: The fraction of retrieved documents that are relevant to a search.

Recall: The fraction of documents that are relevant to a search that are successfully retrieved.

Regular Expression: A concise and flexible means to match strings of text, such as particular characters, words, or patterns of characters.

Stemming: The reduction of a word to its root form.

Suspicious activity: Any activity performed by an individual that could warrant further investigation.

Topic Analysis: The process of analyzing a text for its central topic.

Warez: Any copyrighted works that could be distributed illegally over the Internet.

1.5. Assumptions

The assumptions of the study include:

- It is assumed that any user of this tool does not know a channel or channels to being their investigation.
- The users of IRC will be using the English language.

1.6. Limitations

The limitations of this study include:

- The chat network for the deployment of the framework will be on Internet Relay Chat.
- To mimic a real life scenario, a single topic will be selected in order to narrow down the keyword and topic analysis, to be determined during the actual testing.
- Password protected rooms will be skipped in the crawler module.
- In the crawler module, channels that have under a certain number of users will not be looked through.

1.7. Delimitations

The following delimitations are being made:

- While file sharing is common on IRC, the option for file sharing will not be implemented in this version.
- The Locator Module will not be implemented in this version.

1.8. Summary

This chapter provided an overview of the research project, the scope and significance of the study, as well as the definitions, assumptions, limitations, and delimitations of the study. Also, definitions of terms used in the study are given.

CHAPTER 2. LITERATURE REVIEW

This chapter intends to give an overview of literature related to the study of Internet Relay Chat and investigation frameworks.

2.1. Introduction

Internet Relay Chat (IRC) founds its origin in the late 1980's and is still used today as a large scale communication and file sharing tool. "While the user base for IRC has dropped recently, it is still being used by many different groups ranging from legitimate companies, groups playing "Dungeons and Dragons", warez groups, prank groups, hacktavist groups, illegal information brokers, and terrorists" (Michels, 2011, p. 3). These groups are still active today and still perform their legal and illegal activities. This section will look at a brief history of IRC, an analysis of crime on IRC, and some of the current frameworks and analyses on which this tool will be developed.

2.2. History of Internet Relay Chat

The late 1980s saw the development of one of the first internet chat networks. Jarkko Oikarinen was working at the University of Oulu in Finland when he decided to replace a program used on a Bulletin Board System (BBS) frequented by Oikarinen and his friends. His idea was to have multiple users have multiple users communicate with each other at the same time across the network, dubbed synchronous conferencing (Oikarinen, 2008). After writing the initial code, Oikarinen and his friends tested their initial server with about twenty clients with success. The code was released from the University of Oulu which prompted servers being created at Helsinki University, Tampere University, University of Denver, and Oregon State University. Within the first year of code being release, there were forty servers online worldwide (Stenberg, 2011).

"Throughout many network splits and attempts at standardization, IRC would continue to grow in the 1990s, becoming engrained into the news media starting in 1991" (Michels, 2011, p. 6). Two major conflicts would bring IRC to the forefront of new media. These conflicts were the Gulf War and the August Coup in Russia. During the Gulf War, family members could communicate through IRC around the world (Stenberg, 2011). Also, live reports from the war were sent to news agencies through IRC servers. The influx of people and new media stressed the servers greatly. In the August Coup, an attempted coup by the members of the Soviet Union's government on Mikhail Gorbachev led to a media blackout in Russia. News was given worldwide through IRC of both of the events. The logs on both of these events are still available today (University of North Carolina at Chapel Hill, 1994).

While the number of users has declined since the 2000s, there are still a large number of users on IRC performing a large number of services. Some of the largest servers in the IRC networks still serve over thirty thousand users daily. The top four servers still serve over fifty thousand users worldwide per day (Michels, 2011). The total number of users reached a maximum of 764,621 users spread over 4,579 networks in 2012 (Gelhausen, 2012).

2.3. Crime on Internet Relay Chat Overview

The channels of IRC, while used for their original purpose of synchronous communication, became a playground for illegal activities. These illegal activities fall into certain categories: file sharing, hacking & hacktavist groups, illegal information marketing, and terrorism in both the cyber realm and the physical realm.

A large quantity of the traffic on IRC has to deal with file sharing (Mutton, 2004). Though replaced by new file sharing options like BitTorrent and P2P Networks, IRC still has a large file sharing community, sharing all forms of files from music to software, called "warez." "Warez" is a term short for computer software and usually refers to any copyrighted works that could be distributed illegally (Basamanowicz & Bouchard, 2011). The distribution of software is usually done by a bot that joins channels and publically promotes what that bot has to offer others. Table 2.1 shows the results of searching through the channel lists for any channel name containing the word "warez."

Table 2.1. Channels and Avg. Users in "warez" channels (Michels, 2011)

	Total # of Channels with "Warez" in channel name	Average # of users/room
EFnet	28	8
IRCnet	1	17
Quakenet	0	0
Undernet	45	20

IRCnet and Quakenet completely prohibit any type of warez sharing, while EFnet and Undernet both lack that rule. This information was found by just searching for obvious channel names that would contain the word "warez". Delving deeper into the channels and looking at their descriptions rather than just channel names reveals many other channels with many pieces of warez being shared under the noses of the administrators of those servers.

A study by Paul Mutton analyzed illegal activity on IRC (Mutton, 2004). Mutton ran an IRC client for thirty-six hours in sixty of the largest channels on IRC looking specifically for what he considered illegal activity. Upon entering these channels, a similar message appeared in each channel, somewhat supporting the fact the users in each channel were undertaking illegal activity in that channel. It was found that many of the messages followed the same format:

> If you are affiliated with any government, police, ANTI-Piracy Group, RIAA, MPAA, FBI, movie production company/distribution company or related groups you are violating code 431.322.12 of the Internet Privacy Act signed by Bill Clinton in 1995, therefore you CANNOT threaten our ISP(s), person(s) or company storing these files and cannot prosecute anyone. And you must LEAVE NOW. (Mutton, 2004)

The interesting fact here is that the law mentioned, the Internet Privacy Act, is a fictitious law that was created to help try to give some protection to those operators of web sites and channels that shared illegal data (Healey, 2002). Mutton's analysis found that around 99.9% of IRC traffic in the top sixty channels at that time was dealing with illegal file sharing. He also claims that it is unfair to claim that all of the traffic and conversations that occur on IRC are illegal and that many people are ruining the image of IRC with their actions (Mutton, 2004).

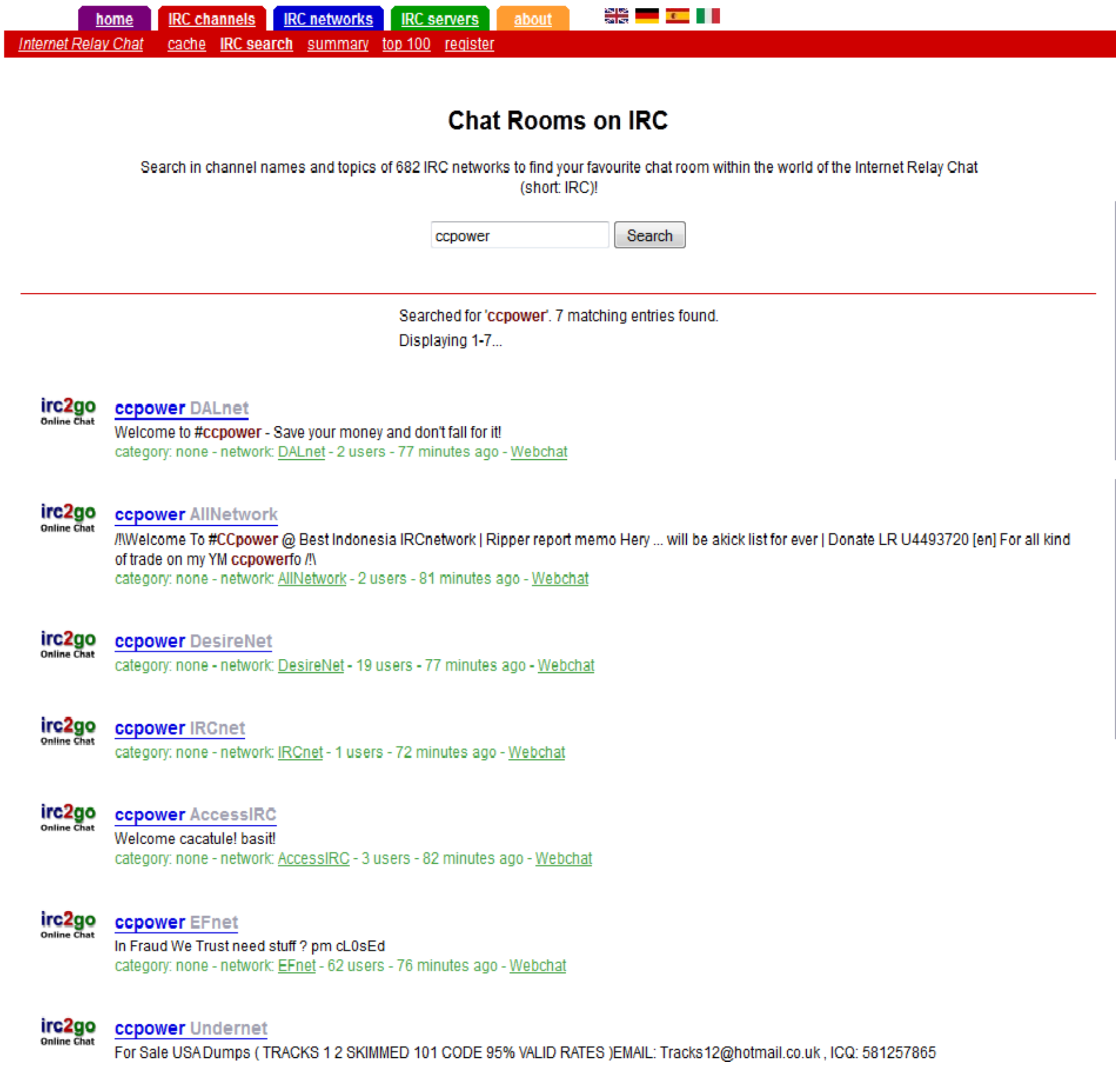
IRC has been and is still used as a staging ground for the activities of groups such as Anonymous, LulzSec, and various other hacking and hacktavist groups. These groups use IRC as one of their meeting grounds to discuss and plan out the attacks they carry out. These groups tend to have a decentralized power structure and no real leading authority figures but act as a communal whole when the time arises (Anonymous Analytics).

An example of their use of IRC came in September 2010 when the group Anonymous began "Operation Payback". Their attacks consisted of distributed denial of service attacks against groups that were trying to shut down sites such as The Pirate Bay and Wikileaks. An investigation into those attacks led to the arrest of a teenager who was one of the operators of the IRC channels of the group Anonymous. Through leaked logs of conversations, it was shown that this teenager helped in the attacks on a selection of websites (Erensto, 2010).

In a similar situation, the IRC logs of those who helped in the attacks undertaken by the group LulzSec in early 2011 were uploaded to the Internet in the Summer of 2011. This log show the dates and times of the attacks planned as well as the showed some of the inner workings of the group itself (Arthur & Gallagher, 2011). The logs shows the reaction of members of the group after their biggest attacks on the Nintendo Corporation, Public Broadcasting Service, InfraGard (a FBI affiliate), and eventually their attacks on Sony which garnered them much media attention. This series of attacks led fallout among the members, with many of them quitting and the log showing the steps that the members took to protect themselves. This information as well as those who fell out of favor with the group helped lead investigators to the arrest of some of the hackers responsible (Eimiller, 2011).

The examples above show that IRC is still a playground for hacking groups with agendas. IRC for them is a hangout that few realize exist anymore. It is the cool place to hang out for any member, from script kiddie to those who find new flaws to exploit. It is their meeting place. It is a safe haven for them. Their power structure somewhat leads to some member being scorned which brings about the opening needed to find those responsible.

IRC has been used for an extended period of time to distribute various types of illegal information: stolen credit card numbers, child pornography, social security numbers, identities, etc. In the height of the use of IRC, the group The Honeypot Project submitted a paper in the “Know Your Enemy” series entitled “Profile: Automated Credit Card Fraud (Spitzer, 2003). The group found IRC channels where users were trading stolen credit card information and shares details about how exactly the users traded the information, providing common channel naming practices, bot structures, common commands, etc. Interestingly enough, some of these channels are still in use today and can be found in some channel server listings such as in Figure 2.1.



Chat Rooms on IRC

Search in channel names and topics of 682 IRC networks to find your favourite chat room within the world of the Internet Relay Chat (short: IRC)!

ccpower

Searched for 'ccpower'. 7 matching entries found.
Displaying 1-7...

ccpower DALnet
Welcome to #ccpower - Save your money and don't fall for it!
category: none - network: DALnet - 2 users - 77 minutes ago - Webchat

ccpower AllNetwork
/!\Welcome To #CCpower @ Best Indonesia IRCnetwork | Ripper report memo Hery ... will be akick list for ever | Donate LR U4493720 [en] For all kind of trade on my YM ccpowerfo /!\
category: none - network: AllNetwork - 2 users - 81 minutes ago - Webchat

ccpower DesireNet
category: none - network: DesireNet - 19 users - 77 minutes ago - Webchat

ccpower IRCnet
category: none - network: IRCnet - 1 users - 72 minutes ago - Webchat

ccpower AccessIRC
Welcome cacatule! basit!
category: none - network: AccessIRC - 3 users - 82 minutes ago - Webchat

ccpower EFnet
In Fraud We Trust need stuff ? pm cL0sEd
category: none - network: EFnet - 62 users - 76 minutes ago - Webchat

ccpower Undernet
For Sale USA Dumps (TRACKS 1 2 SKIMMED 101 CODE 95% VALID RATES)EMAIL: Tracks12@hotmail.co.uk , ICQ: 581257865

Figure 2.1. Channel search for common credit card trading channels (Michels, 2011)

IRC is also used as the market of stolen identities. Jared Shurtz, a blogger for TopTenReviews.com, looked at the market for identity theft and found that "once your identity has been stolen, cybercriminals access an invitation-only Internet Relay Chat site with around 100,000 other cybercriminals and begin auctioning off your identity – and you are part of their online trick or treat" (Shurtz, 2011). A single identity with identification such as name, address, phone number, social security number, etc. goes on sale for about ten dollar per identity. Gaining access to these channels requires the trust of one of the members already in, so finding those responsible for this illegal activity can be a time-consuming exercise (Huston & Miller, 2010).

Terrorists have used IRC servers and channels in the past as a recruiting ground and meeting place. In the book “The Art of Intrusion” famed hacker Kevin D. Mitnick and William L. Simon tells the tale of two young hackers named Comrade and ne0h (Mitnick, 2004). These hackers were used to hack into private and government systems by a supposed terrorist using the name Khalid Ibrahim in the late 1990s.

Khalid began by observing the hacking scene on IRC in the late 1990s using the promise of money and the challenges he thought up to bring ne0h and Comrade into his ploy. He first gave the two some easy starting challenge and then moving the hackers on to targets such as Lockheed Martin and Boeing (Mitnick, 2005). Khalid played the hackers’ egos against them, prompting them with more and more challenges to overcome. This led to the targeting of SIPRNET, a large network of computers used by the United States Department of Defense as well as the United States Department of State (Mitnick, 2005). The situation became all too real for the two hackers when Indian Airlines Flight IC-814 was hijacked by Pakistani terrorists who had ties with the Taliban. Khalid told the hackers and members of ne0h’s hacking group that they were responsible for the hijacking and that Khalid himself was involved, threatening to kill the hackers if they reported him.

An interesting question can be brought up: If Khalid was able to do this in the height of IRC use and the beginning of social media and instant messaging, how easy would it be able to do today? Mitnick claims:

> The combination of determined terrorists and fearless kid hackers could be disastrous for this country. This episode left me wondering how many other Khalids are out there recruiting kids (or even unpatriotic adults with hacking skills) and who hunger after money, personal recognition, or the satisfaction of successfully achieving difficult tasks. The post- Khalid recruiters may be more secretive and not as easy to identify. (Mitnick, 2005)

With the amount of connectivity and with the onset of newer and more secretive forms of communications, the fears of Mitnick now seem like they can happen with great ease. "Due to the widespread use of forums and social media today, it may be easier than ever for the Khalids out on the Internet to find those who would help them both knowingly and unknowingly" (Michels, 2011).

While also being used as a communications tool, IRC has been and is still being used for attacking systems through what can be referred to as cyber-terrorism. The most popular form of this is the botnet, a collection of computers that are connected together and used most often for malicious purposes (Kola, 2008). In the early 2000's, a large number of these botnets were run through IRC networks. The infected computer will connect to an IRC network and sit in a channel, waiting for the controller of the botnet to issue commands. The Storm botnet which plagued the Internet in 2007 had an extremely large number of computers under its command (Francia, 2007). These botnets serve many goals for their masters. The Honeypot Project categorized the uses of botnet under ten categories:

1. Distributed denial of service attacks (DDoS)
2. Spamming
3. Sniffing traffic
4. Key logging
5. Spreading new malware
6. Installing advertisement add-ons and browser helper object
7. Google AdSense abuse
8. Attacking IRC chat networks
9. Manipulating online polls/games
10. Mass identity theft (Bacher et al., 2008)

The number of botnets itself has increased, but the number of IRC controlled botnets has been decreasing with more advanced botnets arriving (e.g. HTTP, mobile). The question remains: why are there still IRC botnets around? The answer is that the botnets must still be being used by parties and making them money (Santorelli, 2010).

2.4. Investigative Frameworks, Architectures, and Current Implementations

There exist a few frameworks for the investigation of chat networks and from there, specifically IRC. One such framework created by Dugald A. Brown discusses an architecture for an automated IRC investigation tool called the Dugald Automated

Investigation Tool or DAIT. Brown looks to reduce the man hours used an in investigation by automating some of the processes of the investigation (Brown, 2007). His architecture is broken down into five modules: the collection module, the analysis module, the storage module, the alert module, and the locator module. Each module works together to help support an investigator in an investigation.

The Collection Module performs the basic functions of a client, sitting in on a channel and listening to conversations according to the IRC Protocol document. This module then takes any messages received, sends an unedited copy of the message to the storage module, then parses the messages for dates, usernames, etc. and sends that data to the analysis module.

The Storage module stores the parsed data from a message into a database. Brown breaks down the database using nine tables to hold all the necessary information:

1. The Session table – records all data pertaining to the network, channel, or private message received.
2. The Handles table – records the username and nickname of users.
3. The Users table – records the logon information of each user.
4. The Message table – Contains each message as well as a link to the user in the Users table as well as the Handles table.
5. The Time table – records when a user logs on or off.
6. The Links table – records any hyperlinks that appear in a message
7. The LinkMsg table – relates hyperlinks to the messages in which they appeared.

8. The ActionList table – shows any interactions between users for the use of social behavior analysis.
9. The Interactions table – records the message where an action took place as well as who performed the action and who received the action.

The Analysis Module is responsible for checking if a crime is being referenced in the messages. Brown uses two types of analysis: keyword analysis and database analysis. The keyword analysis looks for specific keywords (single words), key phrases (multiple words), and regular expressions (for checking for credit card numbers in Brown's example). The database analysis will try to analyze the messages in the database with respect to all other messages in the database to find possible connections in the messages. Brown goes on to discuss three algorithms useful to the database analysis: Naïve Bayes, K-Nearest Neighbor, and Support Vector Machines.

The Alert Module takes the data from the analysis module and sends any discovery to the appropriate law enforcement official. The Alert Module specifically uses email and cell phone messages to alert the appropriate individual to a suspicious message.

The Locator Module takes any user data if available from for a suspicious message and attempts to locate the user. Brown calls for the use of the WHOIS protocol which can look up the location and contact information for any server or IP connected to the Internet. Brown goes on to say that this module may not give any usable information.

Brown's analyses primarily use keyword analysis on the incoming messages as well as analyzing the database for possible connections between people. Regular expressions were used to also indicate criminal activities such as credit card fraud and can be expanded especially in dealing with any numerical data (Brown, 2007). Keyword analysis is a type of analysis that is easily done in real time and has been used by other researchers in the past. The accuracy of this analysis comes down to the list of keywords and its robustness. More keywords in the list will give more possible hits in an investigation while also raising the possibility of a false positive result. There needs to be some other sort of contextual analysis to better understand how the keyword is being used. Part of Speech Tagging (POST) is helpful here. POST helps to break down the sentence into its grammatical parts and categorizing the information (Stanford Natural Language Processing Group, 2012). The number of false positive hits from keyword analysis may be lessened if combined with POST.

In an attempt to create a tool that performs live data analysis, Daniel Cooper created a tool that captured IRC bound packets for inspection, performing live data-analysis on captured messages using keyword analysis as well as topic detection (Cooper, 2011). Cooper mentions in his results that both keyword analysis and topic detection produced false positives and false negatives but gave no definitive performance metrics for his analysis.

The notion of topic analysis has been researched before. Algorithms such as K-Nearest Neighbor, Naïve Bayes, and Support Vector Machines have been used for the categorizing of topics (Elnahrawy, 2002). Associative classification is added on as well

(Dong, Hui, & He, 2006). Cue-phrase analysis and classis frequency analysis has also been used to determine topics from messages (Gainaru, Dumitrescu, & Trausan-Matu, 2010). The Classic Frequency analysis reads a message in a chat line by line and word by words performing frequency analysis. After this is done, only the most used words are kept and then stemmed, combining words that have the same root or are synonyms. This reduces the set of words and the results are the topics in that section of the text.

These frameworks also assume that the investigator has does not need to search the network for any other suspicious activities. When most servers easily have more than a thousand channels (Gelhausen, 2012) searching through each channel on a server can be time consuming for an investigator (Houston & Miller, 2010). An investigator could go through as many channels as possible but that would be time consuming and ill-advised. Other than going through each channel manually, there is no way to accurately know what is going on in each channel, except with the use of automation.

CHAPTER 3. FRAMEWORK AND METHODOLOGY

This chapter looks at how the Dugald Automated Investigation Tool (DAIT) architecture was used to implement the idea of the IRC Data Gathering Tool (IRCDGT) and how the analysis module was be expanded to use other types of textual analyses.

3.1. Methodology

The DAIT features five sections: the client module, the analysis module, the storage module, the alert module, and the locator module. Each of these modules except for the locator module was implemented according to their specifications in the DAIT architecture with some changes to create the IRCDGT.

The program was developed on a PC using 4.00 GB of RAM, using an AMD Phenom II X4 945 3.00 GHz Processor running Windows 7 Ultimate 32-bit edition. The project was implemented in C# due to the ease of implementation as well as coding aspects. Each module corresponds to a section of code for the program.

Various open-source tools were also used to facilitate various parts of the program not easily coded: the SharpNLP Project, a collection of natural language processing tools, the Porter Stemming algorithm, and NHunspell, a spell checking, hyphenation, and thesaurus available for the .NET languages. SharpNLP and NHunspell are used available for use for academic research under the GNU Lesser General Public License (LGPLv3). The Porter Stemming algorithm was coded by the specifications of the original paper by Martin Porter (Porter, 1980).

The client module performed the basic requirements of the client such as connecting to the server, joining channels, sending and receiving messages, and handling private messages. The messages were displayed to the user and then sent for analysis. The client also handled receiving multiple messages from multiple channels at the same time. The client module received commands from another user in the same channel. The client joined a control channel to facilitate this while another client fed commands to it. The basic client was implemented using various tutorials online.

The Analysis module performed real-time analysis of the incoming chat messages. As messages were received (this includes private messages as well), three different types of analyses were performed on the messages which are explained in the next section. Also, to facilitate dynamic keyword selection, the analysis module looked through the storage module searching for keywords that are used frequently. In the course of a conversation (the length of which was determined by the investigator), if a word appeared to dominate the conversation (again, the number of times the word was used was up to the investigator), it was saved as a possible new keyword to be searched for in subsequent searches.

The Storage module was implemented using a simple Excel file to facilitate an actual database but followed the DAIT architecture. The architecture called for storing every message in the stream in the database. This implementation created a log of all messages and saved all messages to that log for any post-mortem chat analysis. Only a certain amount of messages were kept in the database to help facilitate the topic analysis. The rest of the messages were placed in the log for later use.

The Alert module alerted the investigator to a possible suspicious message if one appeared. This was done through the use of email, using a Gmail account to send periodic updates to the investigator. Alerting occurred over a set period of time, usually around every ten minutes if any suspicious messages appeared. The time period could be changed by the investigator as well.

In order to facilitate the issues brought up in the analysis of the framework a final module was implemented which is the Crawler module. This module consisted of two parts: the Topic Crawler and the Channel crawler, each providing similar results.

The Topic Crawler went through the listing of each channel and its description, performing keyword analysis on the description of each channel. This was done to help find channels where suspicious activities from the point of view of the investigator were being undertaken.

The Channel Crawler is similar to the Topic Crawler. The Channel Crawler attempted to go through channels marked as suspicious during the Topic Crawler phase on the server observing the conversations in each channel. It stayed in the channel for a short time listening to messages and performing keyword and topic analysis in order to find channels of suspicious activity. Upon finding a keyword or topic in a message, the channel was flagged for review for the investigator and the crawler continued on to another channel.

3.2. Keyword Analysis

Two types of analyses were examined to try to find which would work the best in a live deployment scenario. Keyword analysis was first examined. The data from the analysis was recorded in a table such as Table 3.1:

Table 3.1 Example results table

	Positive	Negative	Total
Keyword/Topic present	A	B	A+B
Keyword/Topic not present	C	D	C+D
Total	A+C	B+D	A+B+C+D

Area A corresponds to the true positive result, when a keyword is correctly found. Any message containing a keyword that was not identified as suspicious when it is supposed to be will be considered a false negative result, corresponding to a result in area B. Any message containing a keyword that was incorrectly identified as suspicious was placed as a false positive result, corresponding to a result in area C. Area D corresponded to the true negative result, when a keyword is not found. The precision, recall, and accuracy rates were then computed.

$$\text{Precision} = A / (A+B)$$

$$\text{Recall} = A / (A+C)$$

$$\text{Accuracy} = (A + D) / (A+B+C+D)$$

In information retrieval, precision is the fraction of retrieved documents that is relevant to a search, recall is the fraction of documents that are relevant to a search that are successfully retrieved, and accuracy is the proportion of all true results in a search. High recall means that most of the relevant results were returned. High precision means that more relevant than irreverent results were returned.

Next, Keyword analysis was implemented with POST categories attached to each word. For example, looking for the keyword "hacking" as a verb or an adverb instead of just the instance of the word. This analysis was carried out in the same way as the basic keyword analysis. The POST tags are divided up into thirty-six different categories located in Table 3.2.

Table 3.2 Parts of Speech Tags

Tag	Description	Tag	Description	Tag	Description	Tag	Description
CC	Coordinating Conjunction	LS	List item marker	PRP$	Possessive pronoun	VBD	Verb, past tense
CD	Cardinal Number	MD	Modal	RB	Adverb	VBG	Verb, gerund or present participle
DT	Determiner	NN	Noun, singular or mass	RBR	Adverb, comparative	VBN	Verb, past participle
EX	Existential there	NNS	Noun, plural	RBS	Adverb, superlative	VBP	Verb, non-3rd person singular present
FW	Foreign Word	NNP	Proper Noun, singular	RP	Particle	VBZ	Verb, 3rd person singular present
IN	Preposition or subordinating conjunction	NNPS	Proper Noun, plural	SYM	Symbol	WDT	Wh-determiner
JJ	Adjective	PDT	Pre-determiner	TO	To	WP	Wh-pronoun
JJR	Adjective, comparative	POS	Possessive ending	UH	Interjection	WP$	Possessive wh-pronoun
JJS	Adjective, superlative	PRP	Personal Pronoun	VB	Verb, base form	WRB	Wh-adverb

Both the keyword analysis and the keyword & POST analysis were carried out using a simple keyword list made of words about the operating system Ubuntu as well as exploits for the system. This analysis was used in the topic and channel crawlers as well as in normal chatting. The keyword list was comprised of five words and included in the list was the part of speech attached to that word. The list of keywords is shown in Table 3.3

Table 3.3 – Keywords

Keyword	**Part of Speech tag**
Ubuntu	NN
Firmware	NN
Attack	NN or VB
Linux	NN
Hack	NN or VB

After being used to in the topic crawler and channel crawler step, the results of the analyses was saved to an Excel file which contained the channel name, the topic of the channel, and the keyword(s) that were found. The list was then parsed manually to find the performance rates mentioned above, separating the true results from the false results.

The hypothesis for this section of the research is that the keyword analysis with the POST categories would outperform the normal keyword analysis in precision, recall, and accuracy. This was due to the fact that keyword analysis with POST categories would return more specialized results in the analyses.

3.3. Real-Time Topic Analysis

Next, Topic Analysis was implemented. The classic frequency method used by Gainaru, Dumitrescu, & Trausan-Matu (2010) was implemented due to the fact that this type of analysis was easily converted to run well in a live-data acquisition tool and it did not require any lengthy calculations. Figure 3.1 shows the process of topic analysis.

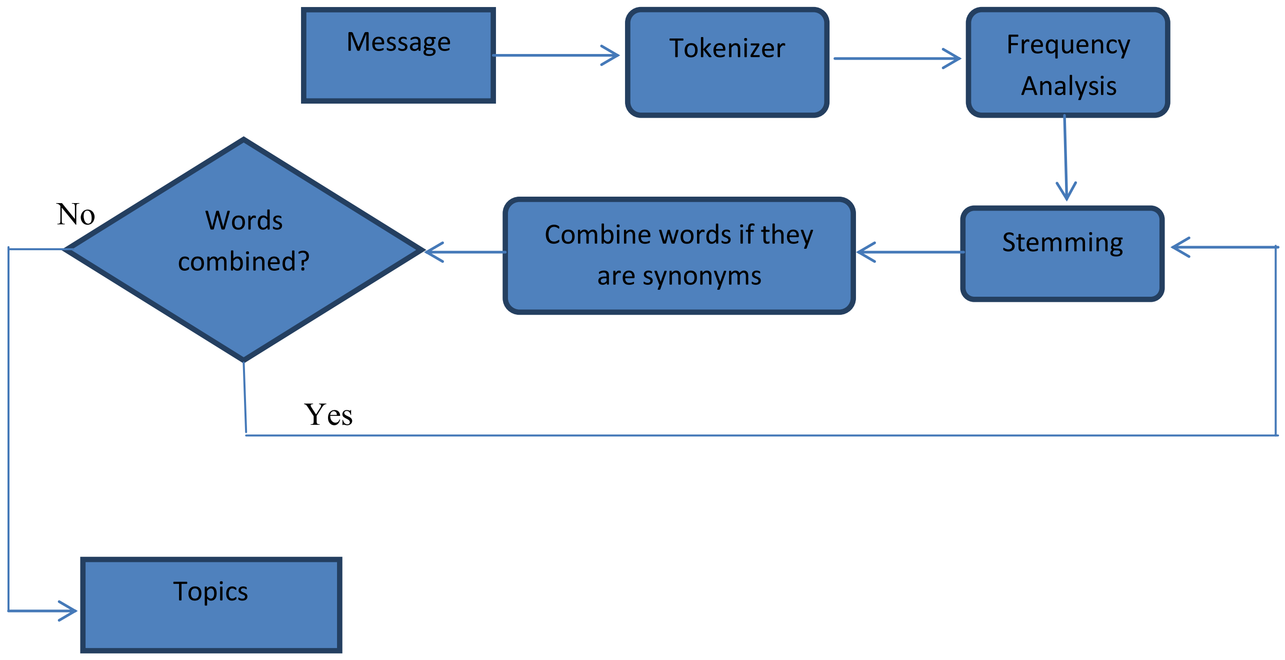


Figure 3.1 – Topic Analysis Flowchart

Most topic analyses are done on a chat log post-mortem. This implementation used a workaround for this by using a certain number of archived messages in order to simulate the post mortem status needed for topic analysis. The analysis module did not perform the topic analysis until a certain number of chat messages were reached, which was called the threshold X. Once that threshold was met, topic analysis was performed using the last X messages to try to find the topics in the conversation. From that point on, each message was subject to topic analysis using the last X messages received. The threshold used was thirty messages for this research. This novel approach was tested to see if changes in the topic could be seen in real time. This required the use of a word stemmer to bring the word to its root form as well as a thesaurus to find synonyms for the words in order to combine them to find the general topics.

CHAPTER 4. RESULTS AND FINDINGS

This was a research experiment to see how keyword analysis, keyword analysis with POST categories, and topic analysis performs in a real-time data acquisition. The research also looked to reduce the number of hours needed for an investigator to be in front of a computer during an investigation on Internet Relay Chat. The main result was the Internet Relay Chat Data Gathering Tool or IRCDGT, a tool that helps provide automated monitoring for IRC channels.

4.1. Implementation

The IRCDGT tool was implemented as described as above with the addition of the Crawler module. Each module was created as a different method in C# using multiple threads to facilitate real-time analysis and performs their duties independent of each other. The program still functions as a normal IRC Client and has the ability to join channels, receive messages, etc. The analyses were run as a message was received and was then stored in a log. The POST tagged messages were outputted to the user as well.

The alert module sent an email out to the investigator every ten minutes. The time in-between messages could be changed by the investigator and eventually moved to sending emails every thirty minutes. This email contains a message digest of any suspicious messages they may have come up as well as the current topics.

The crawler module was tested to see if it was actually possible to perform the analyses in real time and deduct time needed for in investigator to actually be in front of a computer. As the program logged on to the IRC Server, in this case irc.freenode.org, the crawler downloaded the channel list and began to parse through it using both the keyword analysis and the keyword & POST analysis, the results of which are explained in the next section. Any channel that was found containing any of the keywords or combinations of keywords was saved to a list when was then used by the channel crawler.

The channel crawler used the list generated by the topic crawler as the set of channels to be observed. In its initial implementation, the crawler joins the first four channels in the list, spreading out the time for the join command to the crawler does not appear to be flooding the network. Any password protected channel was skipped. The crawler stays in each channel for just over minute logging all messages sent from those channels. Once that minute was up, the crawler left those channels and proceeded to the next four channels until the list is completely was run through. At the end of the list the messages in each channel are sent for analysis and a report is generated on all the suspect channels from those messages showing any suspect messages and the topics that go with them.

Initially the channel crawler did not work due to the join command being sent too rapidly. The server thought the crawler was trying to flood the network with commands and stopped any subsequent join commands for a short while. This was fixed by making the crawler stop for a few seconds before joining the next channel. The server provides a maximum number of channels that one user can join and the program can join up to that amount. The amount of time spent in a channel can also be changed to suit the needs of the investigator.

The crawler module as well as the alert module provides a an investigator with a way to not have to spend so many dedicated hours in front of the monitor looking at the chat. As a test, the crawler module was run overnight on a set of fifty false channels staying in each channel for a minute at four channels per run. Bots were placed in specific channels and would replay the message "Ubuntu chat network, exploits for all." Looking through the fifty channels took just over twelve minutes in total and reported back on the messages given as well as the topic.

4.2. Keyword Analysis Results

Each iteration of the keyword list was run against the topic list of irc.freenode.org. This server was chosen due to its large channel size as well as its connection to the users of the Ubuntu operating system. Retrieving the channel list from the server was the most time consuming part of the research. The channel list would be downloaded every time the analyses were run, taking upwards of 10-17 seconds for the download to complete. The analyses on the topic descriptions took around 18 seconds to complete. The analyses on regular incoming messages were performed un just under a tenth of a second. These time metrics were started just before their respective actions began and ended as soon as they were finished.

The total number of channels found using the keyword list was recorded for each iteration. The analyses looked for any combination of keywords after the first iteration, i.e. on the second iteration, the crawler would return any channel with the keywords “Ubuntu”, “firmware”, or any combination of the two.

Table 4.1 – Avg. channel return list

# of Keywords	**Channels found under Keyword**	**Channels found under Keyword & POST**	**Total Channels**
1	314	46	12276
2	333	59	12275
3	327	62	11517
4	934	340	11484
5	1174	369	11497

Table 4.1 shows how many channels were returned were return These results provided a reduction of between 87%-99% of channels to be searched. Doing this manually would be extremely time consuming to an investigator. The topic lists generated from the analyses were saved and then parsed manually to find the precision, recall, and accuracy rates.

Under one keyword:

Table 4.2 – Keyword Analysis Results with one keyword

Basic Keyword Analysis	**Positive**	**Negative**
Keyword present	260	53
Keyword not present	0	11962
Precision =.8307		
Recall = 1		
Accuracy =.99		

The basic keyword searches for all ten iterations contained no false negative results as seen in Table 4.2. This is due to the fact the channel list was parsed and every instance of the keyword was found and returned as a possible channel to review. This does not take into account the fact that channels can be created/removed after the channel list is downloaded. This automatically makes the recall value for the basic keyword analysis 1.

Table 4.3 – Keyword & POST Analysis Results with one keyword

Keyword & POST Analysis	**Positive**	**Negative**
Keyword/Tag present	39	7
Keyword/Tag not present	221	12005
Precision =.81		
Recall =.15		
Accuracy = .98		

The Keyword & POST analysis did contain false negative results as seen in Table 4.3. This was due to the part of speech tagging analyzing the topic for a channel and not finding the correct tag for the keyword whereas in the basic keyword analysis the channel was a true positive result. This continues out through the other iterations as show in Tables 4.4 through 4.11.

Under two keywords:

Table 4.4 – Keyword Analysis Results with two keywords

Basic Keyword Analysis	**Positive**	**Negative**
Keyword present	273	60
Keyword not present	0	11943
Precision =.8222		
Recall = 1		
Accuracy =.99		

Table 4.5 – Keyword & POST Analysis Results with two keywords

Keyword & POST Analysis	**Positive**	**Negative**
Keyword/Tag present	59	17
Keyword/Tag not present	234	11975
Precision =.77		
Recall = .20		
Accuracy = .98		

Under three keywords:

Table 4.6 – Keyword Analysis Results with three keywords

Basic Keyword Analysis	**Positive**	**Negative**
Keyword present	252	77
Keyword not present	0	11188
Precision = .76		
Recall = 1		
Accuracy =.99		

Table 4.7 – Keyword & POST Analysis Results with three keywords

Keyword & POST Analysis	**Positive**	**Negative**
Keyword/Tag present	40	20
Keyword/Tag not present	253	11204
Precision =.66		
Recall = .13		
Accuracy = .98		

Under four keywords:

Table 4.8 – Keyword Analysis Results with four keywords

Basic Keyword Analysis	**Positive**	**Negative**
Keyword present	262	672
Keyword not present	0	10550
Precision = .28		
Recall = 1		
Accuracy =.98		

Table 4.9 – Keyword & POST Analysis Results with four keywords

Keyword & POST Analysis	**Positive**	**Negative**
Keyword/Tag present	43	297
Keyword/Tag not present	263	11484
Precision = .12		
Recall = .14		
Accuracy = .99		

Under five keywords:

Table 4.10 – Keyword Analysis Results with five keywords

Basic Keyword Analysis	**Positive**	**Negative**
Keyword present	396	672778
Keyword not present	0	10323
Precision = .33		
Recall = 1		
Accuracy =.93		

Table 4.11 – Keyword & POST Analysis Results with five keywords

Keyword & POST Analysis	**Positive**	**Negative**
Keyword/Tag present	68	303
Keyword/Tag not present	406	10720
Precision = .18		
Recall = .14		
Accuracy = .93		

The large jump in numbers from the third keyword to the fourth keyword was due to the fact that many channels descriptions contained the word "linux" in their descriptions. Few of the channels fit into the category of a channel to be looked at though.

Statistical analysis of the results was performed on the data by the use of a t-test. Microsoft Excel was used to carry out the test. Recall was tested first, then precision. The hypothesis for the recall value is shown below:

Recall

H_0: The Keyword analysis would outperform Keyword & POST analysis in terms of recall.

H_a: The Keyword analysis would not outperform Keyword & POST analysis in terms of recall.

Due to keyword analysis always having a recall value of 1, it was easily shown that keyword analysis outperformed keyword analysis with POST. The t-test for precision was performed in the same manner. The hypothesis are recorded below and results are shown in Table 4.12:

Precision

H_0: The Keyword analysis would outperform Keyword & POST analysis in terms of precision.

H_a: The Keyword analysis would not outperform Keyword & POST analysis in terms of precision.

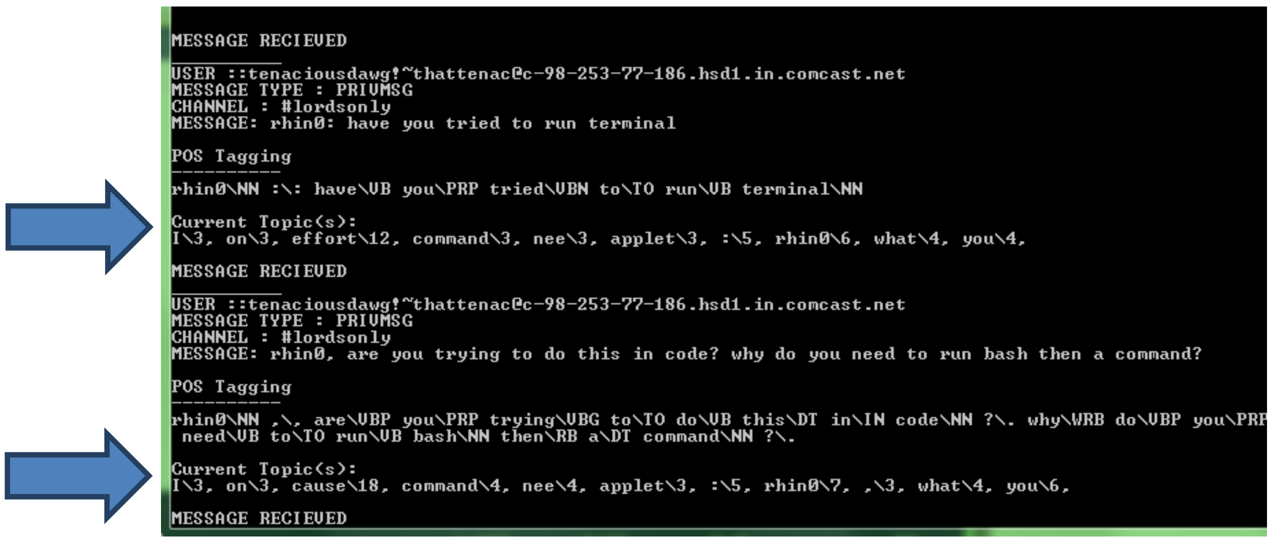
Table 4.12 Precision T-Test

# of Keywords	**Precision for normal keyword**	**Precision for keyword & POST**	**T-Value**	**P-Value**
1	.83	.81	.46	0.66
2	.82	.77		
3	.76	.66		
4	.28	.12		
5	.33	.18		
Means	.60	.51		
Standard Deviation	.27	.34		

There was no significant difference in precision between the normal Keyword Analysis (M=.60, SD=.27) and Keyword Analysis with POST (M=.51, SD=.34); t(8)= .46, p = 0.66.

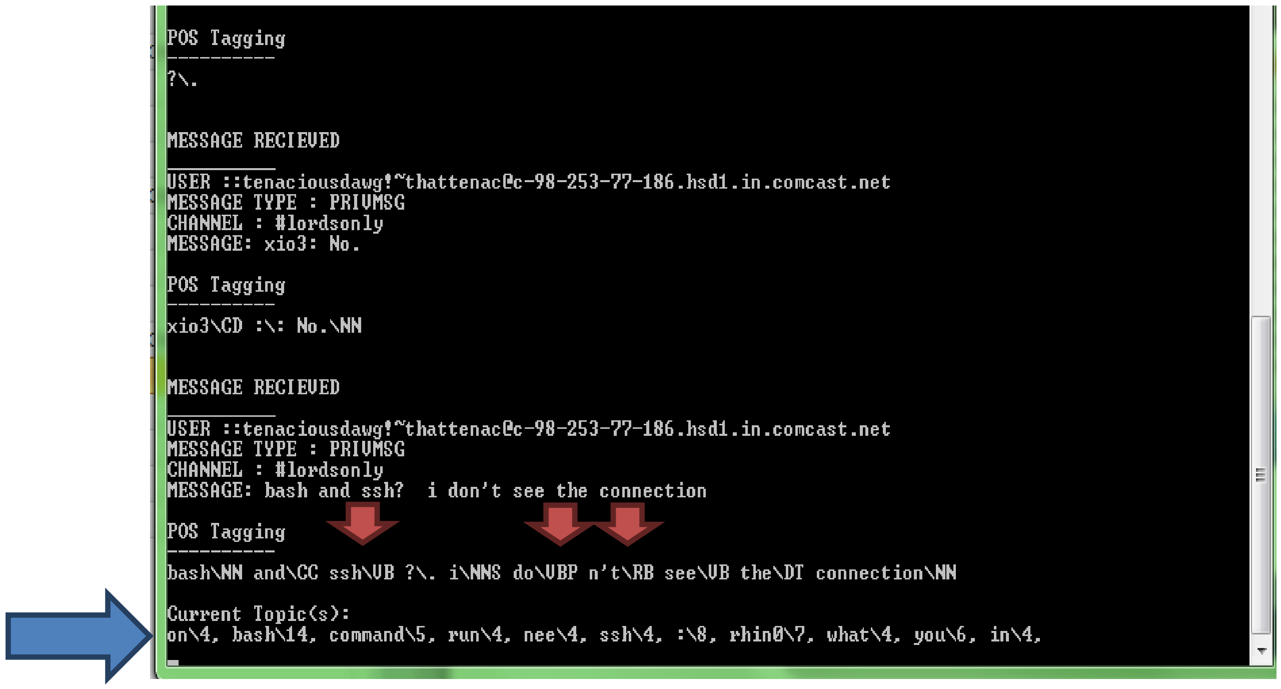
4.3. Topic Analysis Results

As was explained in section three, the topic analysis section was created and run. The threshold value worked in near real time, since there had to be small log of messages in order for this algorithm to work. Once that threshold was reached the topic for the last X messages including the last message was outputted to the user. The POST tagging was also used here to separate out certain types of words that would not make sense as the topic, such as symbols. The threshold value could be changed by the investigator as well. This analysis was tested on two different logs, a public log from the #ubuntu channel on irc.freenode.org as seen in Figure 4.1 and 4.2 and the IRC log from the hacking group LulzSec that was put onto the internet in the summer of 2011.



```
MESSAGE RECIEVED
__________
USER ::tenaciousdawg!~thattenac@c-98-253-77-186.hsd1.in.comcast.net
MESSAGE TYPE : PRIVMSG
CHANNEL : #lordsonly
MESSAGE: rhin0: have you tried to run terminal

POS Tagging
----------
rhin0\NN :\: have\VB you\PRP tried\VBN to\TO run\VB terminal\NN

Current Topic(s):
I\3, on\3, effort\12, command\3, nee\3, applet\3, :\5, rhin0\6, what\4, you\4,

MESSAGE RECIEVED
__________
USER ::tenaciousdawg!~thattenac@c-98-253-77-186.hsd1.in.comcast.net
MESSAGE TYPE : PRIVMSG
CHANNEL : #lordsonly
MESSAGE: rhin0, are you trying to do this in code? why do you need to run bash then a command?

POS Tagging
----------
rhin0\NN ,\, are\VBP you\PRP trying\VBG to\TO do\VB this\DT in\IN code\NN ?\. why\WRB do\VBP you\PRP
 need\VB to\TO run\VB bash\NN then\RB a\DT command\NN ?\.

Current Topic(s):
I\3, on\3, cause\18, command\4, nee\4, applet\3, :\5, rhin0\7, ,\3, what\4, you\6,

MESSAGE RECIEVED
```

Figure 4.1 – Topic Analysis on #ubuntu log, part 1

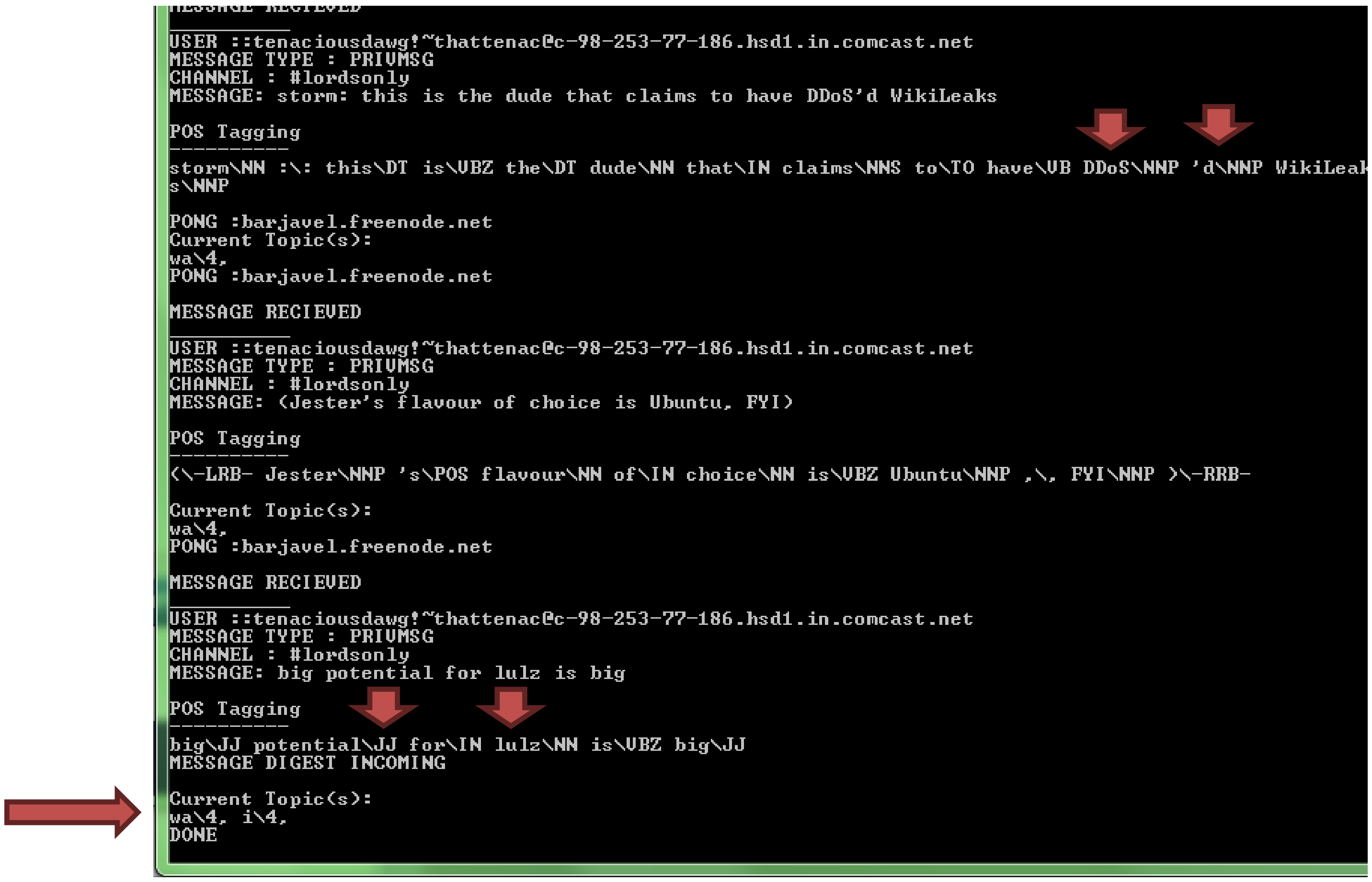


```
POS Tagging
----------
?\.


MESSAGE RECIEVED
________
USER ::tenaciousdawg!~thattenac@c-98-253-77-186.hsd1.in.comcast.net
MESSAGE TYPE : PRIVMSG
CHANNEL : #lordsonly
MESSAGE: xio3: No.

POS Tagging
----------
xio3\CD :\: No.\NN


MESSAGE RECIEVED
________
USER ::tenaciousdawg!~thattenac@c-98-253-77-186.hsd1.in.comcast.net
MESSAGE TYPE : PRIVMSG
CHANNEL : #lordsonly
MESSAGE: bash and ssh?  i don't see the connection

POS Tagging
----------
bash\NN and\CC ssh\VB ?\. i\NNS do\VBP n't\RB see\VB the\DT connection\NN

Current Topic(s):
on\4, bash\14, command\5, run\4, nee\4, ssh\4, :\8, rhin0\7, what\4, you\6, in\4,
```

Figure 4.2 – Topic Analysis on #ubuntu log, part 2

Over the course of time on the #ubuntu channel, after the threshold is of thirty messages was passed, the topic was updated after each message, indicated by the blue arrow. The POST tags were correctly most of the time on the sentence, with a few manageable errors shown by the red arrow above. The topic analysis correctly identified one of the topics in the above conversation, the Linux command "bash" and "ssh", as well as one user talking about how he is going to use it in a web applet, another topic that appeared. Many other words and symbols appear such as the word on, run, ":", what, you, etc., clogging up the topic list. Here the POST tags and the stemmer don't have much issue with the Internet Language here. This is not the case for the other log.

The log from the LulzSec leak did not have as much success as the earlier log. The topics were more spread out over the course of the conversation and POST tagger did not work as intended, becoming confused on some of the terms commonly used on the internet (lol, wtf, etc.) and tagging many of the words wrong. This type of error in tagging occurred when dealing with commands that were discussed in the conversation or the use of internet lingo and memes.



```
USER ::tenaciousdawg!~thattenac@c-98-253-77-186.hsd1.in.comcast.net
MESSAGE TYPE : PRIVMSG
CHANNEL : #lordsonly
MESSAGE: storm: this is the dude that claims to have DDoS'd WikiLeaks

POS Tagging
----------
storm\NN :\: this\DT is\VBZ the\DT dude\NN that\IN claims\NNS to\TO have\VB DDoS\NNP 'd\NNP WikiLeak
s\NNP

PONG :barjavel.freenode.net
Current Topic(s):
wa\4,
PONG :barjavel.freenode.net

MESSAGE RECIEVED
----------
USER ::tenaciousdawg!~thattenac@c-98-253-77-186.hsd1.in.comcast.net
MESSAGE TYPE : PRIVMSG
CHANNEL : #lordsonly
MESSAGE: (Jester's flavour of choice is Ubuntu, FYI)

POS Tagging
----------
(\-LRB- Jester\NNP 's\POS flavour\NN of\IN choice\NN is\VBZ Ubuntu\NNP ,\, FYI\NNP )\-RRB-

Current Topic(s):
wa\4,
PONG :barjavel.freenode.net

MESSAGE RECIEVED
----------
USER ::tenaciousdawg!~thattenac@c-98-253-77-186.hsd1.in.comcast.net
MESSAGE TYPE : PRIVMSG
CHANNEL : #lordsonly
MESSAGE: big potential for lulz is big

POS Tagging
----------
big\JJ potential\JJ for\IN lulz\NN is\VBZ big\JJ
MESSAGE DIGEST INCOMING

Current Topic(s):
wa\4, i\4,
DONE
```

Figure 4.3 – Topic analysis on LulzSec IRC log

The topics gathered from this log were the words "I" and "wa" above as shown in Figure 4.3. While not technically wrong since the use of the word "I" was used during the log, this gives us no idea as to who "I" is referring to or what is the actual topic at hand. "Wa" is not a word at all, but is a product of the stemmer reducing the word “was” to “wa”. Once again, the POST tagger did not handle some of the internet lingo correctly.

CHAPTER 5. DISCUSSION

5.1. Keyword Analysis

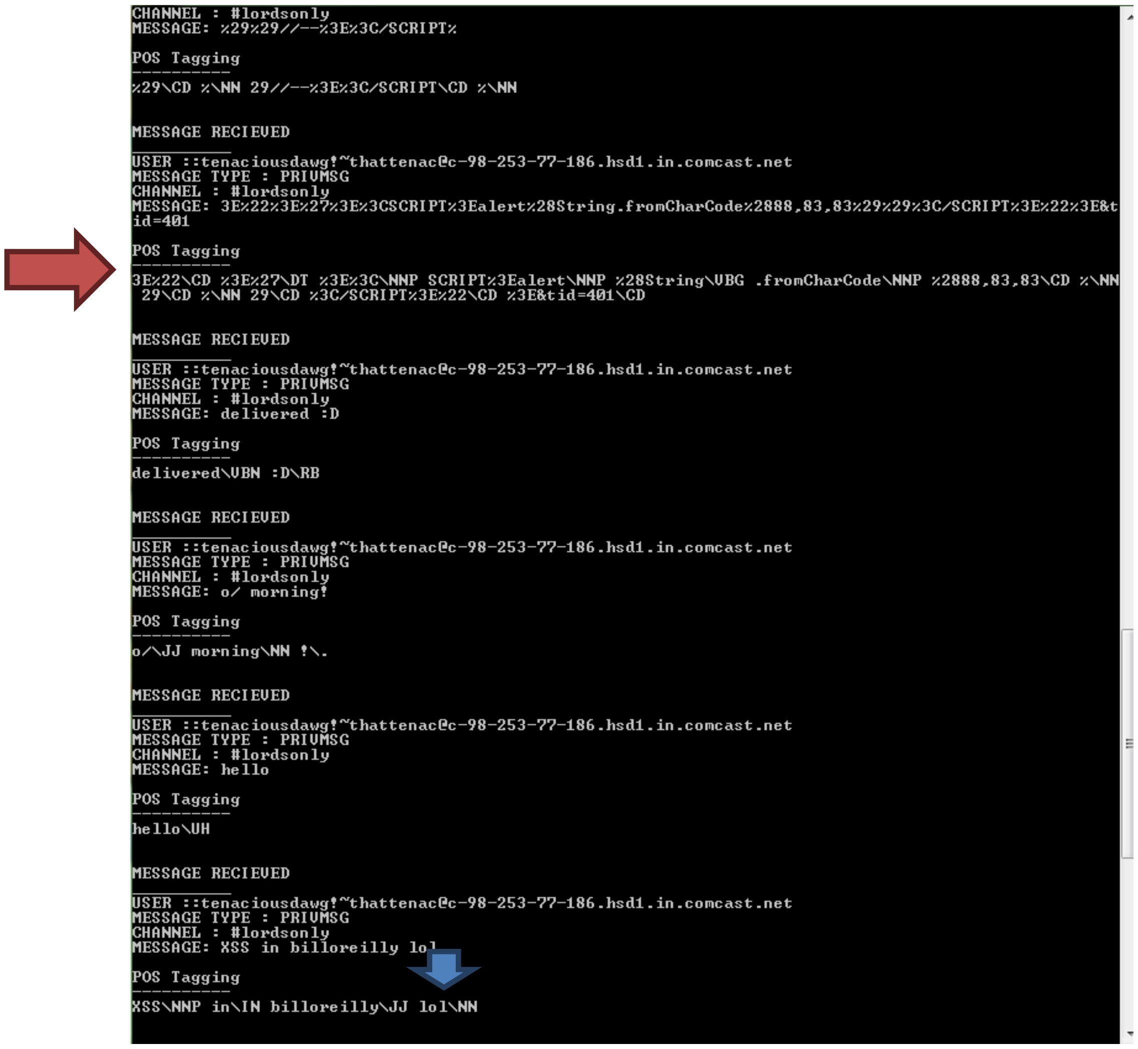
The normal keyword analysis statistically outperformed the keyword & POST analysis in recall while there was no significant difference between the two analyses in precision. The POST tags made the results more restrictive for that analysis and actually kept out channels that could be searched, lowering the number of false positive results but bringing about more false negative results. Recall stayed at 1 for the normal keyword analysis because it always returned all relevant results to that keyword search. The precision scores were close with both analyses with normal keyword analysis staying just higher than keyword & POST. This is once again due to the fact of the more restrictive search results that the keyword & POST analysis brings about.

Something to keep in mind here is that all of this analysis is just on the textual data contained by the channel on the server. The topic may be a good indicator as to what is going on in the channel, but users can change the topic to something more inconspicuous. The crawler module helps to find out what exactly is going on in a suspicious channel, but users not wanting their conversations to be found have a myriad of options to help them stay anonymous. Operators can set passwords to their channels, make their channels invite only, set a limit on the number of users, make the channel secret or private, or all of the above if they wanted to. Other than knowing the password to the room or obtaining server admin rights, it will be very hard to actually know what is going on. Under the assumption that the investigator has no channel to begin an investigation, this tool provides a basic starting area for an investigation.

5.2. Natural Language Processing Issues

There seems to be some issues regarding the internet lingo and natural language processing. The SharpNLP library had trouble dealing with the use of internet terms, commands, and basic internet speech. The distinction to make here is that the language spoken on the Internet is not English, but a subset of English. The stemmer also had issues with the Internet language reducing some words that should not have been reduced.

The language of the Internet is a subset of English and can be difficult to understand at times. Most written and spoken English messages follow set grammatical rules and are comprised of full sentences (subject, verb, object), are usually comprised of one tense (past, present, or future), and contains correct punctuation. Any message in the Internet language is hardly ever a full sentence (lol), can combine multiple tenses into one sentence (I can has cheeseburger), may not have any punctuation, may replace letters with numbers or symbols (1337), or may not make any sense all. Typos run rampant and often become a normal part of the language (teh) and acronyms run rampant. One might say that this language is not natural at all but a twisted version of English.



```
CHANNEL : #lordsonly
MESSAGE: %29%29//--%3E%3C/SCRIPT%

POS Tagging
----------
%29\CD %\NN 29//--%3E%3C/SCRIPT\CD %\NN


MESSAGE RECIEVED
__________
USER ::tenaciousdawg!~thattenac@c-98-253-77-186.hsd1.in.comcast.net
MESSAGE TYPE : PRIVMSG
CHANNEL : #lordsonly
MESSAGE: 3E%22%3E%27%3E%3CSCRIPT%3Ealert%28String.fromCharCode%2888,83,83%29%29%3C/SCRIPT%3E%22%3E&t
id=401

POS Tagging
----------
3E%22\CD %3E%27\DT %3E%3C\NNP SCRIPT%3Ealert\NNP %28String\VBG .fromCharCode\NNP %2888,83,83\CD %\NN
 29\CD %\NN 29\CD %3C/SCRIPT%3E%22\CD %3E&tid=401\CD


MESSAGE RECIEVED
__________
USER ::tenaciousdawg!~thattenac@c-98-253-77-186.hsd1.in.comcast.net
MESSAGE TYPE : PRIVMSG
CHANNEL : #lordsonly
MESSAGE: delivered :D

POS Tagging
----------
delivered\VBN :D\RB


MESSAGE RECIEVED
__________
USER ::tenaciousdawg!~thattenac@c-98-253-77-186.hsd1.in.comcast.net
MESSAGE TYPE : PRIVMSG
CHANNEL : #lordsonly
MESSAGE: o/ morning!

POS Tagging
----------
o/\JJ morning\NN !\.


MESSAGE RECIEVED
__________
USER ::tenaciousdawg!~thattenac@c-98-253-77-186.hsd1.in.comcast.net
MESSAGE TYPE : PRIVMSG
CHANNEL : #lordsonly
MESSAGE: hello

POS Tagging
----------
hello\UH


MESSAGE RECIEVED
__________
USER ::tenaciousdawg!~thattenac@c-98-253-77-186.hsd1.in.comcast.net
MESSAGE TYPE : PRIVMSG
CHANNEL : #lordsonly
MESSAGE: XSS in billoreilly lol

POS Tagging
----------
XSS\NNP in\IN billoreilly\JJ lol\NN
```

Figure 5.1 – Example of Internet Language Issues

Figure 5.1 shows an example of this.   The red arrow points to an http document that was broken up when tokenizing the sentence, a function of the SharpNLP library here. This should be considered as one large object, not broken up into smaller pieces. The blue arrow shows part of the POST tags for the word “lol”. Is “lol” really being used as a noun here? It seems to be used as a verb here. The fact that is can be questioned means something is amiss. This type of error was not uncommon throughout this research.

In order to properly understand that language of the internet, there needs to be more work done in the linguistic patterns of the Internet language. Finding those patterns, if they exist, could greatly affect natural language processing for the better. Any program that uses natural language processing (web searches, keyword searches, etc.) could benefit from this knowledge. The libraries used in this research could not handle some of the internet language and produced errors. While some of these errors can be handled easily, such as expanding acronyms such as "lol" out, there will still be a greater need to expand on the language of the internet. The continued use of natural language processing on the any textual document that contains the internet language will be needed due to the ever changing world of the Internet and its users.

5.3. Topic Analysis

While the approach for finding the topic worked, the results were not all that spectacular. There were two reasons for the lack of substantial results, one being the stemmer developed and the other being the thesaurus. The stemmer did work extremely well, but reduced some words when it should not have, i.e. happy changed to happi. The stemmer cannot achieve perfection. While extra rules may balance it out, it still will make errors (Porter, 2006). There are newer versions of this stemmer out now, and the use of them may future versions on this tool better. These changes to words led to entirely new words showing in the topic analysis.

The use of the thesaurus brought about some issues as well. While not the fault of the actual thesaurus, the list of synonyms brought about issues when trying to determine the actual topic. The POST tags were initially used to try to find in what context was the word being used. This did not work due to the fact that the POST was having issues dealing with Internet lingo and so many of the terms encountered did not have any synonyms to compare with. The meanings of the word would get lost in the synonyms of the other words and reappear in the topics as something completely different. This point brings back the topic of the Internet Language. Most modern dictionaries do not have any definitions, meanings, or synonyms for words that are specific to the Internet Language. There needs to be a change way one looks at the language to include this growing subset of English.

5.4. Implications for Investigators

The analyses used in this took should be the first step for any type of textual analysis in dealing with internet chat networks. The automation of these keyword searches can greatly reduce the number of man hours needed in front of a screen, leaving time for other activities, jobs, or whatever needs to be done. These analyses can help with the needle in a haystack problem that occurs on IRC. Only this time, we are not searching through a haystack, it is more of a hay field.

Investigator is a board term here that is not just meant to be a law enforcement official. This tool is not meant specifically for law enforcement, though this can be used by law enforcement. A company trying to find new flaws or exploits can scour networks to try to find those flaws or exploits or a normal person can use this to find a channel to talk about their hobbies. The hardest part of using a tool like this knowing where to begin searching for suspicious activity. If you know what you are interested in, the keyword search provides a decent starting place to begin your search.

5.5. Future Work

There are many ways to take this research forward. There are a few bugs dealing with making the whole program multi-threaded safe. Ways to improve the keyword analysis should be sought as well. Luckily, the analysis should always provide high precision and accuracy due to the fact that you will get every channel for your keyword no matter what. The improvement here is to get access to even more channels. The best way around that is to become a server admin on a server. The server admin can see invisible users and has the ability to see secret and private channels and can add them to the list of channels to be searched.

In dealing with the topic analysis, other types of topic analysis should be analyzed to see if they can run in real-time or in this case as close to real time as possible. Instead of the unsupervised classification used here, can a supervised approach be used in near-real time using classification techniques? There are many techniques out there to use. Algorithms such as K-Nearest Neighbor, Naïve Bayes, and Support Vector Machines have been used for the categorizing of topics (Elnahrawy, 2002). Associative classification can be used as well (Dong, Hui, & He, 2006).

There also needs to be a move toward a better understanding of the Internet Language in order to keep natural language processing libraries up to date. There exist other natural language processing libraries such as Princeton's WordNet (Princeton University) or the Natural Language Toolkit which can be tested against collections of messages using the Internet language. The creation of a language processor for this Internet language may be needed for future work if the other libraries cannot hold up.

The Locator module should be implemented, but only for those in a law enforcement background. Unfortunately, the data in the locator module might not give an investigator anything of value. The use of software such as Tor can anonymize the information about where a user is, which is usually either their IP address or their hostname. People also use proxies to hide where they really are as well. The implementation of this would not hurt though due to fact that if the correct data is given the module will get a location for a user.

These chats also contain a wealth of user information by just looking at the interactions between users. Social networks can be inferred over multiple users as well as show relationships between those users. The bot pieSpy developed by Paul Mutton draws out the social relationship between users. While the average users using this tool will not have much use for this type of additional tool, this would work well for law enforcement.

A Graphical User Interface is the next step for this tool as well. This can be done easily with the use of C# and the .NET language. This would negate the use of the command channel and make it easier to send actual messages to specific channels.

5.6. Conclusions

IRC is not going anywhere soon and there are still a lot of users on all of those different servers using IRC for whatever goals they have. Finding that needle in the haystack that is an IRC can become much easier now. The reason for this research was twofold: first, to put performance metrics on the simpler types of analyses that are overlooked, and second, to try and reduce the number of man-hours needed in front of a computer by the use of automation. Automation is a means to help an investigation, not necessarily replace one.

LIST OF REFERENCES

LIST OF REFERENCES

Adams, P.H., Martell, C.H. (2008). Topic Detection and Extraction in Chat. Retrieved from http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber= 4597251&url= http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D4597251

Anonymous Analytics. (n.d.) Anonymous Analytics. Retrieved from http://anonanalytics.com/

Arthur, C. & Gallagher, R. (June, 2011). LulzSec IRC leak: the full record. [Web log post]. Retrieved from http://www.guardian.co.uk/technology/2011/jun/24/ lulzsec-irc-leak-the-full-record

Bacher, A., Holz, T., Kotter, M., Wicherski, G. (August, 2008). Know your Enemy: Tracking Botnet. Retrieved from http://www.honeynet.org/papers/bots

Basamanowicz, J., Bouchard, M. (2011). Overcoming the Warez Paradox: Online Piracy Groups and Situational Crime Prevention. *Policy & Internet*: Vol.3: Iss. 2. doi: 10.2202/1944-2866.1125

Brown, D.A., (2007). Architecture for an Automated IRC Investigation Tool, Masters Abstracts Internations. Vol.46, no. 4. 2007

Cooper, D., (2011). Live Data Analysis of Chat Rooms. Retrieved from http://www.tdfcon.org.uk/papers/DC.pdf

Courtney, J., (September, 2011). Skype Usage InfographicL Insightful July 2011 Statistics. Retrieved from http://voiceontheweb.biz/skype-world/skype-ecosystem/video-calling/skype-usage-infographic-insightful-july-2011-statistics/\

Dong, H., Hui, S.C., He, Y. (2006) Structural Analysis of Chat Messages for Topic Detection. Retrieved from http://www.cnts.ua.ac.be/~walter/educational/material/chatMsgAnalysis06.pdf

Eimiller, L. (September, 2011). Member of Hacking Group LulzSec Arrested for June 2011 Intrusion of Sony Pictures Computer Systems. Retrieved from http://www.fbi.gov/losangeles/press-releases/2011/member-of-hacking-group-lulzsec-arrested-for-june-2011-intrusion-of-sony-pictures-computer-systems

Elnahrawy, E. (2002). Log-Based Chat Room Monitoring Using Text Categorization: A Comparative Study. Retrieved from http://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&ved=0CIcBEBYwAA&url=http%3A%2F%2Fciteseerx.ist.psu.edu%2Fviewdoc%2Fdownload%3Fdoi%3D10.1.1.114.2582%26rep%3Drep1%26type%3Dpdf&ei=QCRyT6aVConh0QH35MjcAQ&usg=AFQjCNGiCimWS0iBHxhY_y7IT75Eapxrqw

Engen, V. (2000). The Great Split. Retrieved from http://www.irc.org/history_docs/TheGreatSplit.html

Enersto. (2010). Anonymous' Operation Payback IRC Operator Arrested. [Web log post]. Retrieved from http://torrentfreak.com/anonymous-operation-payback-irc-operator-arrested-101210/

Francia, R. (October, 2007). Storm Word network shrinks to about one-tenth of its former size. Retrieved from http://tech.blorge.com/Structure:%20/2007/10/21/2483/

Gainaru, A., Dumitrescu, S.D., Trausan-Matu, S. (July, 2010). Toolkit for automatic analysis of chat conversations. Retrieved from http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=5509066

Gelhausen, A., (2012). Internet Relay Chat. Retrieved from http://irc.netsplit.de/

Healey, J. (November, 2002). Some Web Sites are Posting a 'Keep out' Sign to Law Enforcement. Retrieved from http://articles.latimes.com/2002/nov/19/business/fi-dvd19

Houston, B., Miller, M. (2010). Beware the Darknet. Retrieved from www.procysive.com/info_wp/Beware_the_Darknet.pdf

Kola, M.K., (2008). Botnets: Overview and Case Study. Retrieved from https://www.mercy.edu/ias/kola.pdf

NHunspell. (2010). NHunspell. Retrieved from http://nhunspell.sourceforge.net/

Michels, M. (December, 2011). Internet Relay Chat: Do We Need To Worry About It?. Retrieved from web.ics.purdue.edu/~michels/IRC.docx

Mitnick, Kevin D. (2005). *The Art of Intrustion: The Real Stories Behind the Exploits of Hackers, Intruders and Deceivers*. Indianapolis, IN: Wiley Publishing.

Mutton, P. (2004). IRC Analysis. Retrieved from http://www.jibble.org/irc-analysis/

Oikarinen, J., (April, 2008). Founding IRC. Retrieved from http://www.mirc.com/jarkko.html

Oikarinen, J., (May, 1993). Internet Relay Chat Protocol. Retrieved from http://www.rfc-editor.org/rfc/rfc1459.txt

Porter, M.F. (1980) An Algorithm for Suffix Stripping. Retrieved from http://tartarus.org/~martin/PorterStemmer/def.txt

Princeton University. (2010). About Wordnet. Retrieved from http://wordnet.princeton.edu

Santorelli, S. (November, 2010). Episode 77: Dead Botnets [Video file]. Retrieved from http://www.youtube.com/watch?v=vYTM9s15yk4

SharpNLP. (December 13, 2006). SharpNLP – open source natural language processing tools. Retrieved from http://sharpnlp.codeplex.com/

Shurtz, J. (October, 2011). Horrifying Ways Cybercriminals Use Your Identity. [Web log post]. Retrieved from http://identity-theft-protection-services-review.toptenreviews.com/horrifying-ways-cybercriminals-use-your-identity.html

Spitzer, L. (June, 2003) Know Your Enemy: Automated Credit Card Fraud. Retrieved from http://www.honeynet.org/papers/profiles

Stanford Natural Language Processing Group. (March, 2012). Stanford Log-linear Part-Of-Speech Tagger. Retrieved from http://nlp.stanford.edu/software/tagger.shtml.

Stenberg, D. (March, 2011). History of IRC. Retrieved from http://daniel.haxx.se/irchistory.html

Ubuntu IRC Logs. (2012). #ubuntu.txt. Retrieved from http://irclogs.ubuntu.com/2012/04/18/%23ubuntu.txt

United States v. Rogelio Jackett, Jr., No. 1:11-cr-00096-AJT (2011).

APPENDIX

APPENDIX

Figure A.1 – Implementation Code

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Net.Sockets;
using System.IO;
using System.Net;
using System.Net.Mail;
using System.Threading;
using System.Windows.Forms;
using OpenNLP.Tools.Tokenize;
using OpenNLP.Tools.PosTagger;
using OpenNLP.Tools.Util;
using NHunspell;
using System.Diagnostics;

struct IRCConfig
{
   public string server;
   public int port;
   public string nick;
   public string name;
   public List<IRCChannel> channels;
   public List<IRCChannel> currentChannels;
   public string modelPath;
   public string actual_server;
   public List<string> keywords;
   public List<keywordPOST> keywordsPOST;
   public List<message> alerts;
}

struct IRCChannel
{
   public string name;
   public string users;
   public int numUsers;
   public string topic;
   public List<message> messages;
```

```
    public int numberOfMessages;
    public string keywordsFound;
    public string currentTopics;
}

struct keywordPOST
{
    public string keyword;
    public List<string> tag;
}

struct message
{
    public string user;
    public string server;
    public string channel;
    public string userMessage;
    public string timeFromUTC;
    public int foundWith;
    public string keywordsFound;
}

struct user
{
    string nickname;
    string username;
}


namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WindowHeight = 70;
            Console.WindowWidth = 100;

            IRCConfig conf = new IRCConfig();
            conf.name = "Insert User Name Here"; //Username
            conf.nick = "Insert Nick Name Here"; //Nickname
            conf.port = 6667;
            conf.server = "Insert Server Name Here"; //Server to Connect To
            conf.channels = new List<IRCChannel>();
            conf.alerts = new List<message>();
            string mModelPath =
System.IO.Path.GetDirectoryName(System.Reflection.Assembly.GetExecutingAssembly().GetN
ame().CodeBase);
            mModelPath = new System.Uri(mModelPath).LocalPath + @"\Models\";
            conf.modelPath = mModelPath;

            new IRCDGT(conf);

            Console.WriteLine("Finished Running");
            Console.ReadLine();
        }
    }

    class IRCDGT
    {
        TcpClient IRCConnection = null;
        public IRCConfig config;
        NetworkStream ns = null;
        StreamReader sr = null;
        StreamWriter sw = null;

        StreamWriter writer;
        EnglishMaximumEntropyTokenizer tokenizer = null;
        EnglishMaximumEntropyPosTagger posTagger = null;
        List<message> messages = null;

        //The message threshold for topic analysis
        const int MESSTHRESH = 20;
        //The time until the next message digest
        const int TIMEUNTILNEXTDIGEST = 15;
        //The maximum number of channels to join during the channel crawling
        const int CHANSTOJOIN = 3;

        DateTime nextCheck = DateTime.Now.AddMinutes(TIMEUNTILNEXTDIGEST);

        public string MAXCHANNEL;

        /*
         * Name: IRCDGT
         * Description: Class that performs all work dealing with IRC Channel
         *
        */
        public IRCDGT(IRCConfig config)
        {
            config.keywords = new List<string>();
            config.currentChannels = new List<IRCChannel>();


            //Read in keyword list
```

```
Console.WriteLine("Reading keyword list\n__________");
try
{
    using (StreamReader sr = new StreamReader("keywords.csv"))
    {
        String line = sr.ReadLine();
        while ((line = sr.ReadLine()) != null)
        {
            string[] split = line.Split(',');
            for (int i = 0; i < split.Length; i++)
            {
                if (!split[i].Equals(""))
                {
                    Console.WriteLine(split[i]);
                    config.keywords.Add(split[i]);
                }
            }
        }
    }
}
catch (Exception e)
{
    Console.WriteLine("The file could not be read:");
    Console.WriteLine(e.Message);
}

//Read in keyword & POST list

Console.WriteLine("\nReading keyword&POST list\n__________");
config.keywordsPOST = new List<keywordPOST>();
try
{
    keywordPOST kPOST;

    using (StreamReader sr = new StreamReader("keywordsPOST.csv"))
    {
        String line = sr.ReadLine();
        while ((line = sr.ReadLine()) != null)
        {
            kPOST = new keywordPOST();
            kPOST.tag = new List<string>();

            string[] split = line.Split(',');
            for (int i = 0; i < split.Length; i++)
            {
                if (i == 0)
                    kPOST.keyword = split[i];
                else
                {
                    if (!split[i].Equals(""))
                    kPOST.tag.Add(split[i]);
                  }

                }

                Console.Write(kPOST.keyword + "\\");
                foreach (string tagged in kPOST.tag)
                  Console.Write(tagged + " ");
                Console.WriteLine();

                config.keywordsPOST.Add(kPOST);
              }
            }
          }
          catch (Exception e)
          {
            Console.WriteLine("The file could not be read:");
            Console.WriteLine(e.Message);
          }


          //Configure the TCP client to connect to the IRC server
          this.config = config;
          try
          {
            IRCConnection = new TcpClient(config.server, config.port);
          }
          catch
          {
            Console.WriteLine("Connection Error");
          }

          try
          {
            DateTime now = DateTime.Now;
            writer = new StreamWriter(config.server + "_" + now.Day + "-" + now.Month + "-" +
now.Year + ".txt");
            messages = new List<message>();
            Console.WriteLine();

            ns = IRCConnection.GetStream();
            sr = new StreamReader(ns);
            sw = new StreamWriter(ns);
            sendData("USER", config.nick + " swivvet.com " + " swivvet.com" + " :" +
config.name);
            sendData("NICK", config.nick);
            IRCActions();
```

```
    }
    catch
    {
        Console.WriteLine("Communication error");
    }
    finally
    {
        if (sr != null)
            sr.Close();
        if (sw != null)
            sw.Close();
        if (ns != null)
            ns.Close();
        if (IRCConnection != null)
            IRCConnection.Close();


        //update all keywords
        using (System.IO.StreamWriter file = new System.IO.StreamWriter("keywords.csv"))
        {
            file.WriteLine("Keyword updated from " + DateTime.Now);

            foreach (string kword in this.config.keywords)
            {
                file.WriteLine(kword);
            }
        }

        writer.Close();
    }

}

/*
 * Name: sendData
 * Description: sends data to the IRC server
 *
 */
public void sendData(string cmd, string param)
{
    if (param == null)
    {
        sw.WriteLine(cmd);
        sw.Flush();
        writer.WriteLine(cmd);
        Console.WriteLine(cmd);
    }
    else
    {
            sw.WriteLine(cmd + " " + param);
            sw.Flush();
            writer.WriteLine(cmd + " " + param);
            Console.WriteLine(cmd + " " + param);
        }
    }

    /*
     * Name: IRCActions
     * Description: recieves data from the server and performs various actions depending
     *             on the data.
     *
     */
    public void IRCActions()
    {
        string[] ex;
        string data;
        bool shouldRun = true;
        bool gotchannels = true;
        bool quittime = false;
        tokenizer = new EnglishMaximumEntropyTokenizer(config.modelPath +
"EnglishTok.nbin");
        posTagger = new EnglishMaximumEntropyPosTagger(config.modelPath +
"EnglishPOS.nbin");


        while (shouldRun)
        {
            if (quittime == true)
            {
                sendData("QUIT", "LATER"); //if the command is quit, send the QUIT command to
the server with a quit message
                shouldRun = false; //turn shouldRun to false - the server will stop sending us data so
trying to read it will not work and result in an error. This stops the loop from running and we will
close off the connections properly
            }
            else
            {

                data = sr.ReadLine();
                char[] charSeparator = new char[] { ' ' };
                ex = data.Split(charSeparator, 5);

                //returns the pong command to a ping request
                if (ex[0] == "PING")
                {
                    config.actual_server = ex[1];
                    sendData("PONG", ex[1]);
                }
```

```
          //grabs the channel list once connected to the server, then move on to other
commands
          else if (gotchannels == false)
          {
            gotchannels = true;
            IRCChannel channel;
            Stopwatch myWatch = new Stopwatch();
            myWatch.Start();
            sendData("LIST", null);

            bool moreChannels = true;

            data = sr.ReadLine();
            Console.WriteLine(data);
            charSeparator = new char[] { ' ' };
            ex = data.Split(charSeparator, 5);


            //shows all users in a channel
            while (!(ex[1].Equals("319") || ex[1].Equals("321") || ex.Equals("322")))
            {
              data = sr.ReadLine();
              Console.WriteLine(data);
              charSeparator = new char[] { ' ' };
              ex = data.Split(charSeparator, 5);
            }

            //keeps reading channel listing until finished
            while (moreChannels == true)
            {
              data = sr.ReadLine();
              //Console.WriteLine(data);
              charSeparator = new char[] { ' ' };
              ex = data.Split(charSeparator, 5);

              channel = new IRCChannel();
              channel.name = ex[3].ToLower();
              channel.users = ex[4].Substring(0, ex[4].IndexOf(' ')).ToLower();
              channel.topic = ex[4].Substring(ex[4].IndexOf(' ')).ToLower();
              config.channels.Add(channel);

              if (ex[1].Equals("323"))
              {
                moreChannels = false;
              }
            }

            //topic crawl in a new thread
            Thread topicCrawler = new Thread(new ThreadStart(TopicCrawl));
```

```
                    topicCrawler.Start();
                    myWatch.Stop();

                    Console.WriteLine("Elapsed Time for Getting List :" +
myWatch.Elapsed.TotalSeconds.ToString());

                    //join the controlling channel
                    sendData("JOIN", "#controller");
                    IRCChannel newChan = new IRCChannel();
                    newChan.name = "#lordsonly";
                    newChan.messages = new List<message>();
                    newChan.numberOfMessages = 0;
                    newChan.numUsers = 0;
                    config.currentChannels.Add(newChan);


                }
                else
                {
                    //Check if time for next message digest
                    if (DateTime.Now > nextCheck)
                    {
                        Thread update = new Thread(latestAlert);
                        update.Start();
                    }

                    //checks for end of message of the day
                    if (ex[1].Equals("376"))
                    {
                        Console.WriteLine(data);
                        Console.WriteLine("End of Messages");
                        config.actual_server = ex[0];
                        gotchannels = false;
                    }
                    //adds user names to channel description
                    else if (ex[1].Equals("353"))
                    {
                        string chan = ex[4].Substring(0, ex[4].IndexOf(' '));

                        for (int i = 0; i < config.currentChannels.Count; i++)
                        {
                            if (chan.Equals(config.currentChannels[i].name))
                            {
                                IRCChannel replacer = config.currentChannels[i];   //change to
nick,username
                                replacer.users = replacer.users + ex[4].Substring(ex[4].IndexOf(' ') + 2);
                                string[] users = replacer.users.Split(' ');

                                for (int j = 0; j < users.Length; j++)
```

```
            {
                if (!users[j].Equals(' '))
                    replacer.numUsers++;
            }

            config.currentChannels.RemoveAt(i);
            config.currentChannels.Insert(i, replacer);
        }
    }

    Console.WriteLine(" USERS in channel " + chan + " : " + ex[4]);
}
else if (ex[1].Equals("372"))  //Displays the server's Message of the Day
{

    Console.WriteLine(data);
}
else if (ex[1] == "PART")
{
    //User quit channel
}
else if (ex[1] == "QUIT")
{
    //user quit server
}
else if (ex[1] == "JOIN") //user joins a channel
{
    string username = ex[0].Substring(0, ex[0].IndexOf("@"));

    for (int i = 0; i < config.currentChannels.Count; i++)
    {
        if (ex[2].Equals(config.currentChannels[i].name))
        {
            IRCChannel replacer = config.currentChannels[i];
            replacer.users = replacer.users + username;
            replacer.numUsers++;

            config.currentChannels.RemoveAt(i);
            config.currentChannels.Insert(i, replacer);
        }
    }
}
else if (ex[1] == "PRIVMSG")  //Deals with any message from a user
{
    Stopwatch myWatch = new Stopwatch();
    myWatch.Start();

    writer.WriteLine(data);
    writer.Flush();
```

```
                    Console.WriteLine("\nMESSAGE RECIEVED\n__________\nUSER :" + ex[0]
+ "\nMESSAGE TYPE : " + ex[1] + "\nCHANNEL : " + ex[2]);
                    string[] tokens;

                    if (ex[3].Equals(":!join") || ex[3].Equals(":!say") || ex[3].Equals(":!quit") ||
ex[3].Equals(":!chancrawl") || ex[3].Equals(":!part"))        //commands first
                    {
                        //Command messages
                        string command = ex[3];

                        switch (command)
                        {
                            case ":!part":
                                sendData("PART", ex[2]);
                                foreach (IRCChannel chan in config.currentChannels)
                                {
                                    if (chan.name.Equals(ex[2]))

config.currentChannels.RemoveAt(config.currentChannels.IndexOf(chan));
                                }

                                break;
                            case ":!chancrawl":
                                channelCrawl();
                                break;
                            case ":!join":
                                sendData("JOIN", ex[4]);
                                IRCChannel newChan = new IRCChannel();
                                newChan.name = ex[4];
                                newChan.messages = new List<message>();
                                newChan.numberOfMessages = 0;
                                newChan.numUsers = 0;
                                config.currentChannels.Add(newChan);
                                break;
                            case ":!say":
                                sendData("PRIVMSG", ex[2] + "  " + ex[4]);
                                break;
                            case ":!quit":
                                sendData("QUIT", ex[4]);
                                shouldRun = false;
                                break;
                        }
                    }
                    else
                    {
                        //Tokenize the message and send for analysis
                        String mess;
                        message saveMe = new message();
                    if (ex.Length < 5)
                    {
                       Console.WriteLine("MESSAGE: " + ex[3].Substring(1) + "\n");
                       mess = ex[3].Substring(1);
                       tokens = tokenizer.Tokenize(ex[3].Substring(1));

                    }
                    else
                    {
                       Console.WriteLine("MESSAGE: " + ex[3].Substring(1) + " " + ex[4] +
"\n");

                       mess = ex[3].Substring(1) + " " + ex[4];
                       tokens = tokenizer.Tokenize(ex[3].Substring(1) + " " + ex[4]);
                    }

                    //Pass 1 - Keyword Analysis

                    saveMe.userMessage = mess;
                    saveMe.foundWith = 0;
                    saveMe.keywordsFound = "";
                    bool kfound = false;
                    bool kPOST = false;

                    foreach (string keyword in config.keywords)
                    {
                       foreach (string tok in tokens)
                       {
                          if (tok.Equals(keyword))
                          {
                             if (!kfound)
                             {
                                saveMe.server = config.server;
                                saveMe.user = ex[0];
                                saveMe.channel = ex[2];
                                saveMe.userMessage = mess;
                                saveMe.timeFromUTC = DateTime.UtcNow.ToLongDateString();
                                saveMe.foundWith = 1;
                                saveMe.keywordsFound = saveMe.keywordsFound + keyword +
",";
                             }

                          }
                       }
                    }

                    //Pass 2 - Keyword analysis with tags

                    string[] tags = Analyze(tokens);
```

```
                foreach (keywordPOST keyword in config.keywordsPOST)
                {
                    foreach (string tagged in keyword.tag)
                    {
                        for (int i = 0; i < tokens.Length; i++)
                        {
                            if (tokens[i].Equals(keyword.keyword) && tagged.Contains(tags[i]))
                            {
                                if (!kPOST)
                                {
                                    saveMe.server = config.server;
                                    saveMe.user = ex[0];
                                    saveMe.channel = ex[2];
                                    saveMe.userMessage = mess;
                                    saveMe.timeFromUTC =
DateTime.UtcNow.ToLongDateString();
                                    saveMe.foundWith = 2;
                                    saveMe.keywordsFound = saveMe.keywordsFound +
keyword.keyword + "\\" + tagged + ",";
                                }
                            }
                        }
                    }
                }

                if (saveMe.foundWith != 0)
                {
                    config.alerts.Add(saveMe);
                }

                //Pass 3 - Dynamic Topic Analysis

                for (int i = 0; i < config.currentChannels.Count; i++)
                {
                    if (ex[2].Equals(config.currentChannels[i].name))
                    {
                        if (config.currentChannels[i].messages.Count > MESSTHRESH)
                        {
                            config.currentChannels[i].messages.RemoveAt(0);
                            config.currentChannels[i].messages.Add(saveMe);
                        }
                        else
                        {

                            IRCChannel replacer = config.currentChannels[i];
                            replacer.messages.Add(saveMe);
                            replacer.numberOfMessages++;

                            config.currentChannels.RemoveAt(i);
```

```
                                config.currentChannels.Insert(i, replacer);
                            }
                        }
                    }

                    for (int i = 0; i < config.currentChannels.Count; i++)
                    {
                        if (config.currentChannels[i].messages.Count >= MESSTHRESH)
                        {

                            Thread topic = new Thread(topicAnalysis);
                            topic.Start();
                        }
                    }
                }

                myWatch.Stop();
                Console.WriteLine("Elapsed Time for Single Message: " +
myWatch.Elapsed.TotalSeconds.ToString() + "\n");
            }
            else
            {
                DateTime now = DateTime.UtcNow;
                Console.Write(now + ": " + data);
                Console.WriteLine();
            }
        }
    }
  }
}

/*
 * Name: Analyze
 * Description: Analyze a message for its POS tags
 *
 */
public string[] Analyze(string [] mess)
{

    string[] tags = posTagger.Tag(mess);
    Console.WriteLine("POS Tagging\n----------");

    for (int l = 0; l < mess.Length; l++)
    {
        Console.Write(mess[l] + "\\" + tags[l] + " ");
    }

    Console.WriteLine();
```

```
        return tags;
    }

    /*
    * Name: mVal
    * Description: computes the m-value of a word.  Used in stemming
    *
    */
    public int mVal(string word)
    {
        bool cons = false;
        bool vowel = false;
        int m = 0;

        for (int i = 0; i < word.Length; i++)
        {
            if (i == 0)
            {
                if (word[i].Equals('a') || word[i].Equals('e') || word[i].Equals('i') || word[i].Equals('o') ||
word[i].Equals('u'))
                    vowel = true;
                else
                    cons = true;
            }
            else
            {
                if (word[i].Equals('a') || word[i].Equals('e') || word[i].Equals('i') || word[i].Equals('o') ||
word[i].Equals('u'))
                {
                    vowel = true;
                    cons = false;
                }
                else if (vowel == true && cons == false)
                {
                    cons = true;
                    vowel = false;
                    m++;
                }
                else
                {
                    cons = true;
                    vowel = false;
                }
            }
        }

        return m;
    }
```

```
/*
* Name: stemmer
* Description: Stems a given word to its root form
*
*/
public string stemmer(string _word)
{
   string word = _word;
   int m = 0;

   m = mVal(word);

   //Step 1a
   if (word.EndsWith("sses"))
      word = word.Replace("sses", "ss");
   else if (word.EndsWith("ies"))
      word = word.Replace("ies", "i");
   else if (word.EndsWith("ss"))
      word = word.Replace("ss", "ss");
   else if (word.EndsWith("s"))
      word = word.Replace("s", "");

   //Step 1b
   string stem;
   bool secondStep = false;
   if (m > 0)
   {
      if (word.EndsWith("eed"))
         word = word.Replace("eed", "ee");
   }

   if (word.EndsWith("ed"))
   {
      stem = word.Replace("ed", "");
      if (stem.Contains('a') || stem.Contains('e') || stem.Contains('i') || stem.Contains('o') ||
stem.Contains('u'))
      {
         word = word.Replace("ed", "");
         secondStep = true;
      }
   }
   else if (word.EndsWith("ing"))
   {
      stem = word.Replace("ing", "");
      if (stem.Contains('a') || stem.Contains('e') || stem.Contains('i') || stem.Contains('o') ||
stem.Contains('u'))
      {
         word = word.Replace("ing", "");
         secondStep = true;
```

```
            }
        }

        if (secondStep)
        {
            m = mVal(word);

            if (word.EndsWith("at"))
                word = word.Replace("at", "ate");
            else if (word.EndsWith("bl"))
                word = word.Replace("bl", "ble");
            else if (word.EndsWith("iz"))
                word = word.Replace("iz", "ize");
            else if (word[word.Length - 1].Equals(word[word.Length - 2]))
            {
                if (!(word[word.Length - 1].Equals('s') || word[word.Length - 1].Equals('l') ||
word[word.Length - 1].Equals('z')))
                    word = word.Substring(0, word.Length - 1);
            }
            else if (m == 1)
            {
                if (word.Length >= 3)
                {
                    if (!(word[word.Length - 3].Equals('a') || word[word.Length - 3].Equals('e') ||
word[word.Length - 3].Equals('i') || word[word.Length - 3].Equals('o') || word[word.Length -
3].Equals('u')))
                    {
                        if (word[word.Length - 2].Equals('a') || word[word.Length - 2].Equals('e') ||
word[word.Length - 2].Equals('i') || word[word.Length - 2].Equals('o') || word[word.Length -
2].Equals('u'))
                        {
                            if (!(word[word.Length - 1].Equals('a') || word[word.Length - 1].Equals('e') ||
word[word.Length - 1].Equals('i') || word[word.Length - 1].Equals('o') || word[word.Length -
1].Equals('u')))
                            {
                                if (!(word[word.Length - 1].Equals('w') || word[word.Length -
1].Equals('x') || word[word.Length - 1].Equals('y')))
                                {
                                    word = word + 'e';
                                }
                            }
                        }
                    }
                }
            }
        }

        //Step 1c
```

```
            if (word.EndsWith("y"))
            {
               stem = word.Replace("y", "");
               if (stem.Contains('a') || stem.Contains('e') || stem.Contains('i') || stem.Contains('o') ||
stem.Contains('u'))
               {
                  word = word.Replace("y", "i");
               }
            }

            //step 2
            m = mVal(word);
            if (m > 0)
            {
               if (word.EndsWith("ational"))
                  word = word.Replace("ational", "ate");
               else if (word.EndsWith("logi"))
                  word = word.Replace("logi", "log");
               else if (word.EndsWith("tional"))
                  word = word.Replace("tional", "tion");
               else if (word.EndsWith("enci"))
                  word = word.Replace("enci", "ence");
               else if (word.EndsWith("anci"))
                  word = word.Replace("anci", "ance");
               else if (word.EndsWith("izer"))
                  word = word.Replace("izer", "ize");
               else if (word.EndsWith("bli"))
                  word = word.Replace("bli", "ble");
               else if (word.EndsWith("alli"))
                  word = word.Replace("alli", "al");
               else if (word.EndsWith("entli"))
                  word = word.Replace("entli", "ent");
               else if (word.EndsWith("eli"))
                  word = word.Replace("eli", "e");
               else if (word.EndsWith("ousli"))
                  word = word.Replace("ousli", "ous");
               else if (word.EndsWith("ization"))
                  word = word.Replace("ization", "ize");
               else if (word.EndsWith("ation"))
                  word = word.Replace("ation", "ate");
               else if (word.EndsWith("ator"))
                  word = word.Replace("ator", "ate");
               else if (word.EndsWith("alism"))
                  word = word.Replace("alism", "al");
               else if (word.EndsWith("iveness"))
                  word = word.Replace("iveness", "ive");
               else if (word.EndsWith("fulness"))
                  word = word.Replace("fulness", "ful");
               else if (word.EndsWith("ousness"))
```

```
        word = word.Replace("ousness", "ous");
    else if (word.EndsWith("aliti"))
        word = word.Replace("aliti", "al");
    else if (word.EndsWith("iviti"))
        word = word.Replace("iviti", "ive");
    else if (word.EndsWith("biliti"))
        word = word.Replace("biliti", "ble");
}

//step 3
m = mVal(word);

if (m > 0)
{
    if (word.EndsWith("icate"))
        word = word.Replace("icate", "ic");
    else if (word.EndsWith("ative"))
        word = word.Replace("ative", "");
    else if (word.EndsWith("alize"))
        word = word.Replace("alize", "al");
    else if (word.EndsWith("iciti"))
        word = word.Replace("iciti", "ic");
    else if (word.EndsWith("ical"))
        word = word.Replace("ical", "ic");
    else if (word.EndsWith("ful"))
        word = word.Replace("ful", "");
    else if (word.EndsWith("ness"))
        word = word.Replace("ness", "");
}

//step 4

m = mVal(word);

if (m > 1)
{
    if (word.EndsWith("al"))
        word = word.Replace("al", "");
    else if (word.EndsWith("ance"))
        word = word.Replace("ance", "");
    else if (word.EndsWith("ence"))
        word = word.Replace("ence", "");
    else if (word.EndsWith("er"))
        word = word.Replace("er", "");
    else if (word.EndsWith("ic"))
        word = word.Replace("ic", "");
    else if (word.EndsWith("able"))
        word = word.Replace("able", "");
    else if (word.EndsWith("ement"))
```

```
                word = word.Replace("ement", "");
            else if (word.EndsWith("ment"))
                word = word.Replace("ment", "");
            else if (word.EndsWith("ent"))
                word = word.Replace("ent", "");
            else if (word.EndsWith("sion"))
                word = word.Replace("sion", "");
            else if (word.EndsWith("tion"))
                word = word.Replace("tion", "");
            else if (word.EndsWith("ou"))
                word = word.Replace("ou", "");
            else if (word.EndsWith("ism"))
                word = word.Replace("ism", "");
            else if (word.EndsWith("ate"))
                word = word.Replace("ate", "");
            else if (word.EndsWith("iti"))
                word = word.Replace("iti", "");
            else if (word.EndsWith("ous"))
                word = word.Replace("ous", "");
            else if (word.EndsWith("tion"))
                word = word.Replace("ive", "");
            else if (word.EndsWith("ive"))
                word = word.Replace("ou", "");
        }

        //Step 5a skipping

        //Step 5b
        m = mVal(word);

        if (m > 1)
        {
            if (word[word.Length - 1].Equals(word[word.Length - 2]))
            {
                if (!word[word.Length - 1].Equals('l'))
                    word = word.Substring(0, word.Length - 1);
            }
        }

        return word;
    }

    /*
    * Name: TopicCrawl
    * Description: performs keyword and keyword & POST analysis on
    * the channel list retrieved earlier.
    *
    */
    public void TopicCrawl()
```

```
   {

      List<IRCChannel> suspectChannels = new List<IRCChannel>();
      List<IRCChannel> suspectChannelsPOST = new List<IRCChannel>();

      int numberofChannels = 0;
      int numberofPOSTChannels = 0;
      string[] tokens;
      string[] tags;
      bool found = false;
      IRCChannel suspectChannel;
      Stopwatch myWatch = new Stopwatch();
      myWatch.Start();


      //Pass 1 - Keyword Analysis
      for (int i = 0; i < config.channels.Count; i++)
      {
         found = false;
         suspectChannel = config.channels[i];

         for (int j = 0; j < config.keywords.Count; j++)
         {
            if (config.channels[i].topic.Contains(config.keywords[j]))
            {
               found = true;
               suspectChannel.keywordsFound = suspectChannel.keywordsFound + "," +
config.keywords[j];
            }
         }

         if (found)
         {
            suspectChannels.Add(suspectChannel);
            numberofChannels++;
         }
      }

      //Pass 2 - Keyword w/ POST
      for (int i = 0; i < config.channels.Count; i++)
      {
         tokens = tokenizer.Tokenize(config.channels[i].topic);  //tokenize
         tags = posTagger.Tag(tokens); //tag

         foreach (keywordPOST keyword in config.keywordsPOST)
         {
            found = false;
            suspectChannel = config.channels[i];
```

```
                foreach (string tagged in keyword.tag)
                {
                    for (int j = 0; j < tokens.Length; j++)
                    {
                        if (tokens[j].Equals(keyword.keyword) && tagged.Contains(tags[j]))
                        {
                            if (!found)
                            {
                                found = true;
                                suspectChannel.keywordsFound = suspectChannel.keywordsFound + "  " +
keyword.keyword + "\\" + tags[j];
                            }
                        }
                    }

                    if (found)
                    {
                        suspectChannelsPOST.Add(suspectChannel);
                        numberofPOSTChannels++;
                    }
                }
            }
        }
        myWatch.Stop();

        Console.WriteLine("Elapsed Time for topic crawling : " +
myWatch.Elapsed.TotalSeconds.ToString());


        MessageBox.Show("Keyword Analysis on Topic Description\r\nFound " +
numberofChannels.ToString() + " channels with keywords in it out of " + config.channels.Count
+
            "\r\n\r\nKeyword Analysis w/ POST on Topic Description\r\nFound " +
numberofPOSTChannels.ToString() + " channels with keywords in it out of " +
config.channels.Count);

        //Output text to file

        using (System.IO.StreamWriter file = new System.IO.StreamWriter(config.server +
"_topicCrawlerAnalysis_" + DateTime.Today.Day + "_" + DateTime.Today.Month + "_" +
DateTime.Now.Year + ".csv"))
        {
            file.WriteLine("Channel Name,, Keyword(s), Topic");

            foreach (IRCChannel chan in suspectChannels)
            {
                file.WriteLine(chan.name + "," + chan.keywordsFound + "," + chan.topic);
            }
        }
```

```
            using (System.IO.StreamWriter file = new System.IO.StreamWriter(config.server +
"_topicCrawlerAnalysisPOST_" + DateTime.Today.Day + "_" + DateTime.Today.Month + "_" +
DateTime.Now.Year + ".csv"))
            {
                file.WriteLine("Channel Name,, Keyword(s), Topic");

                foreach (IRCChannel chan in suspectChannelsPOST)
                {
                    file.WriteLine(chan.name + ",," + chan.keywordsFound + "," + chan.topic);
                }
            }
            Console.WriteLine("CHECKED!");
        }

        /*
        * Name: pingpong
        * Description: automatically sends the PONG command every minute
        *
        */
        public void pingpong()
        {
            while (true)
            {
                sendData("PONG", config.actual_server);
                Thread.Sleep(60000);
            }
        }

        /*
        * Name: latestalert
        * Description: sends a message digest of all suspicious messages
        *
        */
        public void latestAlert()
        {

            Console.WriteLine("MESSAGE DIGEST INCOMING");

            var fromAddress = new MailAddress("insert_from_email_address@here.com",
"investigator");
            var toAddress = new MailAddress("insert_to_email_address@here.com", "investigator");
            const string fromPassword = "insert password here";
            const string subject = "Message Digest";

            string body = "";

            if (config.alerts.Count != 0)
            {
```

```
            foreach (message mess in config.alerts)
            {

               body = body + "\r\n\nUser: " + mess.user + "\r\nServer: " + mess.server +
                  "\r\nChannel: " + mess.channel + "\r\nMessage: " + mess.userMessage +
"\r\nTime: " + mess.timeFromUTC;
            }
         }
         else
         {
            body = body + "No alerts at this present time\n";
         }

         body = body + "\n\nCurrent Topics:\r\n";
         foreach (IRCChannel chan in config.currentChannels)
         {
            body = body + "\r\n" + chan.currentTopics;
         }

         var smtp = new SmtpClient
         {
            Host = "smtp.gmail.com", //change if not using gmail
            Port = 587,
            EnableSsl = true,
            DeliveryMethod = SmtpDeliveryMethod.Network,
            UseDefaultCredentials = false,
            Credentials = new NetworkCredential(fromAddress.Address, fromPassword)
         };

         using (var message = new MailMessage(fromAddress, toAddress)
         {
            Subject = subject,
            Body = body
         })
         {
            smtp.Send(message);
         }

         nextCheck = DateTime.Now.AddMinutes(TIMEUNTILNEXTDIGEST);
         Console.WriteLine("DONE");

      }

      /*
      * Name: topicAnalysis
      * Description: performs topicAnalysis outside of normal chatting.  Used
      * in channel crawling
      *
      */
```

```
    public void topicAnalysis()
    {

        MyThes thesaurus = new MyThes("th_en_us_new.dat");
        List<string> words = new List<string>();
        List<int> freq = new List<int>();

        bool nextWord = false;

        try
        {

            foreach (IRCChannel chan in config.currentChannels)
            {
                if (chan.messages.Count >= MESSTHRESH)
                {
                    foreach (message mess in chan.messages)
                    {
                        string[] tokens = tokenizer.Tokenize(mess.userMessage);
                        string[] tags = posTagger.Tag(tokens);

                        for (int i = 0; i < tokens.Length; i++)
                        {
                            int index = i;

                            if (!(tags[i].Contains("CD") || tags[i].Contains('.') || tags[i].Contains("DT") ||
tags[i].Contains("DT") || tags[i].Contains("LS") || tags[i].Contains("MD") || tags[i].Contains(
"SYM") || tags[i].Contains("TO") || tags[i].Contains("UH")))
                            {
                                if (words.Contains(tokens[i]))
                                {
                                    index = words.IndexOf(tokens[i]);
                                    freq[index] = freq[index] + 1;
                                }
                                else
                                {
                                    if (tokens[i].Length != 1 || tokens[i].Length != 2)
                                    {
                                        words.Add(tokens[i]);
                                        freq.Add(1);
                                    }
                                }
                            }

                        }
                    }//end foreach

                    //get rid of anything under a low frequency - 10th of threshold
                    for (int i = 0; i < freq.Count; i++)
```

```
{
   if (freq[i] <= (MESSTHRESH / 10))
   {
      freq.RemoveAt(i);
      words.RemoveAt(i);
      i--;
   }

   if (freq[i] >= (MESSTHRESH * .25))
   {
      //prompt for addition to keyword list
   }
}

//These next steps repeat until no synonyms can be found

//stem + synonyms
for (int i = 0; i < words.Count; i++)
{
   words[i] = stemmer(words[i]);
}

//look up synonyms
ThesResult tr;
List<string> firstWordMean;
List<string> secWordMean;
bool noSynonyms = false;

while (!noSynonyms)
{
   noSynonyms = true;
   for (int i = 0; i < words.Count; i++)
   {
      for (int j = i + 1; j < words.Count; j++)
      {
         tr = thesaurus.Lookup(words[i]);
         firstWordMean = new List<string>();

         if (tr == null)
         {
            //nothing in the thesaurus, keep checking other words for possible match
on basic word
         }
         else
         {
            //collect all meanings and synonyms

            foreach (ThesMeaning mean in tr.Meanings)
            {
```

```
          foreach (string syn in mean.Synonyms)
          {
             if (!syn.Contains(' '))
                firstWordMean.Add(syn.ToLower());
          }
       }

       secWordMean = new List<string>();

       tr = thesaurus.Lookup(words[j]);
       if (tr == null)
       {
          //nothing in the thesaurus, keep checking other words for possible
match on basic word
       }
       else
       {
          foreach (ThesMeaning mean in tr.Meanings)
          {

             foreach (string syn in mean.Synonyms)
             {
                if (!syn.Contains(' '))
                   secWordMean.Add(syn.ToLower());
             }
          }
       }

       //compare words

       nextWord = false;

       foreach (string secWord in secWordMean)  //checking if basic word is in
synonym or meaning list
       {
          if (words[i].Equals(secWord) && !nextWord)
          {
             //successful match of first word to either meaning or syn of second
word

             words[i] = secWord;
             freq[i] = freq[i] + freq[j];
             words.RemoveAt(j);
             freq.RemoveAt(j);
             j--;

             //Console.WriteLine("Successful Synonym Match 1: " + words[i] +
" " + words[j] + " from " + secWord);
             nextWord = true;
```

```
                                        noSynonyms = false;
                                    }
                                }

                                foreach (string firWord in firstWordMean)
                                {
                                    if (words[j].Equals(firWord) && !nextWord)
                                    {
                                        //successful match of second wordd to either meaning or syn of first
word

                                        words[i] = firWord;
                                        freq[i] = freq[i] + freq[j];
                                        words.RemoveAt(j);
                                        freq.RemoveAt(j);
                                        j--;

                                        //Console.WriteLine("Successful Synonym Match 2: " + words[i] +
" " + words[j] + " from " + firWord);
                                        nextWord = true;
                                        noSynonyms = false;
                                    }
                                }

                                foreach (string firWord in firstWordMean)
                                {
                                    foreach (string secWord in secWordMean)
                                    {
                                        if (firWord.Equals(secWord) && !nextWord)
                                        {
                                            //successful match of one of the first words meanings or
synonyms to either the meaning or syn of second word

                                            words[i] = secWord;
                                            freq[i] = freq[i] + freq[j];
                                            words.RemoveAt(j);
                                            freq.RemoveAt(j);
                                            j--;

                                            //Console.WriteLine("Successful Synonym Match 3: " + words[i]
+ " " + words[j] + " from " + firWord + " to " + secWord);
                                            nextWord = true;
                                            noSynonyms = false;
                                        }
                                    }
                                }
                            }
                        }
                    }
```

```
            }

            Console.WriteLine("Current Topic(s): ");
            writer.Write("Current Topic(s): ");

            IRCChannel chan2 = chan;
            chan2.currentTopics = "";

            for (int i = 0; i < words.Count; i++)
            {
                Console.Write(words[i] + "\\" + freq[i] + ", ");
                writer.Write(words[i] + "\\" + freq[i] + ", ");
                chan2.currentTopics = chan2.currentTopics + words[i] + "\\" + freq[i] + ", ";
            }

            int replace = config.currentChannels.IndexOf(chan);
            config.currentChannels.RemoveAt(replace);
            config.currentChannels.Insert(replace, chan2);

            Console.WriteLine();
            writer.WriteLine();

            //if no synonyms found the noMore = true;

        }
    }
  }
  catch (Exception e)
  {

  }
}

/*
* Name: listenForData
* Description: Listens to the server for all data from the server.  Used
* in channel crawling
*
*/
public void listenForData()
{
  while (true)
  {
    string data = sr.ReadLine();
    char[] charSeparator = new char[] { ' ' };
    string[] ex = data.Split(charSeparator, 5);

    String mess = "";
    String[] tokens = null;
```

```
            message saveMe = new message();


            if (ex.Length == 3 || ex[1].Equals("353") || ex[1].Equals("366"))
            {
              Console.WriteLine(data);
            }
            else if (ex[1].Equals("475"))
            {
              Console.WriteLine("Channel password protected, skipping");
            }
            else
            {
              Console.WriteLine("\nMESSAGE RECIEVED\n__________\nUSER :" + ex[0] +
"\nMESSAGE TYPE : " + ex[1] + "\nCHANNEL : " + ex[2]);
              if (ex.Length < 5)
              {
                //Console.WriteLine("MESSAGE: " + ex[3].Substring(1) + "\n");
                mess = ex[3].Substring(1);
                tokens = tokenizer.Tokenize(ex[3].Substring(1));

              }
              else
              {
                // Console.WriteLine("MESSAGE: " + ex[3].Substring(1) + " " + ex[4] + "\n");
                mess = ex[3].Substring(1) + " " + ex[4];
                tokens = tokenizer.Tokenize(ex[3].Substring(1) + " " + ex[4]);
              }

              //Pass 1 - Keyword Analysis

              saveMe.userMessage = mess;
              saveMe.foundWith = 0;
              bool kfound = false;
              bool kPOST = false;

              foreach (string keyword in config.keywords)
              {
                foreach (string tok in tokens)
                {
                  if (tok.Equals(keyword))
                  {
                    if (!kfound)
                    {
                      saveMe.server = config.server;
                      saveMe.user = ex[0];
                      saveMe.channel = ex[2];
                      saveMe.userMessage = mess;
                      saveMe.timeFromUTC = DateTime.UtcNow.ToLongDateString();
                    saveMe.foundWith = 1;
                }

            }
        }
    }

    //Pass 2 - Keyword analysis with tags

    string[] tags = Analyze(tokens);

    foreach (keywordPOST keyword in config.keywordsPOST)
    {
        foreach (string tagged in keyword.tag)
        {
            for (int i = 0; i < tokens.Length; i++)
            {
                if (tokens[i].Equals(keyword.keyword) && tagged.Contains(tags[i]))
                {
                    if (!kPOST)
                    {
                        saveMe.server = config.server;
                        saveMe.user = ex[0];
                        saveMe.channel = ex[2];
                        saveMe.userMessage = mess;
                        saveMe.timeFromUTC = DateTime.UtcNow.ToLongDateString();
                        saveMe.foundWith = 2;
                    }
                }
            }
        }
    }

    for (int i = 0; i < config.currentChannels.Count; i++)
    {
        if (ex[2].Equals(config.currentChannels[i].name))
        {
            IRCChannel replacer = config.currentChannels[i];
            replacer.messages.Add(saveMe);
            replacer.numberOfMessages++;

            config.currentChannels.RemoveAt(i);
            config.currentChannels.Insert(i, replacer);
        }
    }
  }
 }
}
```

```
/*
* Name: channelCrawl
* Description: Listens to the server for all data from the server.  Used
* in channel crawling
*
*/
public void channelCrawl()
{
   string data;

   Thread pp = new Thread(pingpong);
   pp.Start();

   Thread listen = new Thread(listenForData);
   listen.Start();

   using (System.IO.StreamReader file = new System.IO.StreamReader("channellist.csv"))
   {
      data = file.ReadLine();

      while (!file.EndOfStream)
      {
         string[] channelName = new string[CHANSTOJOIN];
         for (int i = 0; (i < CHANSTOJOIN) && (!file.EndOfStream); i++)
         {
            data = file.ReadLine();
            channelName[i] = data.Substring(0, data.IndexOf(','));
            sendData("JOIN", channelName[i]);
            IRCChannel newChan = new IRCChannel();
            newChan.name = channelName[i]; ;
            newChan.messages = new List<message>();
            newChan.numberOfMessages = 0;
            newChan.numUsers = 0;
            config.currentChannels.Add(newChan);
            Thread.Sleep(2000);
         }

         DateTime end = DateTime.Now.AddMinutes(1);
         while (DateTime.Now < end)
         {

         }

         for (int i = 0; i < CHANSTOJOIN; i++)
         {
            if (channelName[i] != null || channelName[i] != "")
               sendData("PART", channelName[i]);
         }
      }
```

```
            }

            listen.Abort();
            pp.Abort();

            Thread analysis = new Thread(CrawlerAnalysis);
            analysis.Start();
        }

        /*
        * Name: CrawlerAnalysis
        * Description: Performs the analysis on all of the channels observed.
        *
        */
        public void CrawlerAnalysis()
        {
            List<IRCChannel> chans = config.currentChannels;
            MyThes thesaurus = new MyThes("th_en_us_new.dat");
            List<string> words;
            List<int> freq;
            string report = "";
            bool nextWord = false;
            StreamWriter writer = new StreamWriter("channelCralwerAnalysis" +
DateTime.Today.DayOfYear + ".txt");

            foreach (IRCChannel chan in chans)
            {
                words = new List<string>();
                freq = new List<int>();
                writer.WriteLine("-----------\r\nAnalysis of Channel " + chan.name);
                writer.WriteLine("Messages: " + chan.messages.Count);

                foreach (message mess in chan.messages)
                {
                    if (mess.foundWith > 0)
                    {
                        writer.WriteLine("User: " + mess.user + "\r\nServer: " + mess.server +
"\r\nChannel: " + mess.channel);
                        writer.WriteLine("Message: " + mess.userMessage + "\r\nKeywords found: " +
mess.keywordsFound + "\r\nTime: " + mess.timeFromUTC);
                    }

                    string[] tokens = tokenizer.Tokenize(mess.userMessage);
                    string[] tags = posTagger.Tag(tokens);

                    for (int i = 0; i < tokens.Length; i++)
                    {
                        int index = i;
            if (tags[i].Contains("NN") || tags[i].Contains("VB"))
            {
                if (words.Contains(tokens[i]))
                {
                    index = words.IndexOf(tokens[i]);
                    freq[index] = freq[index] + 1;
                }
                else
                {
                    if (tokens[i].Length != 1)
                    {
                        words.Add(tokens[i]);
                        freq.Add(1);
                    }
                }
            }
        }
    }

    writer.WriteLine("\r\n\r\nTopic Analysis for " + chan.name);

    //get rid of anything under certain frequency
    for (int i = 0; i < freq.Count; i++)
    {
        if (freq[i] <= (MESSTHRESH / 10))
        {
            freq.RemoveAt(i);
            words.RemoveAt(i);
            i--;
        }
    }

    //These next steps repeat until no synonyms can be found

    //stem + synonyms
    for (int i = 0; i < words.Count; i++)
    {
        words[i] = stemmer(words[i]);
    }

    //look up synonyms
    ThesResult tr;
    List<string> firstWordMean;
    List<string> secWordMean;

    for (int i = 0; i < words.Count; i++)
    {
        for (int j = i + 1; j < words.Count; j++)
        {
```

```
                    tr = thesaurus.Lookup(words[i]);
                    firstWordMean = new List<string>();

                    if (tr == null)
                    {
                        //nothing in the thesaurus, keep checking other words for possible match on
basic word
                    }
                    else
                    {
                        //collect all meanings and synonyms

                        foreach (ThesMeaning mean in tr.Meanings)
                        {
                            foreach (string syn in mean.Synonyms)
                            {
                                if (!syn.Contains(' '))
                                    firstWordMean.Add(syn.ToLower());
                            }
                        }

                        secWordMean = new List<string>();

                        tr = thesaurus.Lookup(words[j]);
                        if (tr == null)
                        {
                            //nothing in the thesaurus, keep checking other words for possible match on
basic word
                        }
                        else
                        {
                            foreach (ThesMeaning mean in tr.Meanings)
                            {

                                foreach (string syn in mean.Synonyms)
                                {
                                    if (!syn.Contains(' '))
                                        secWordMean.Add(syn.ToLower());
                                }
                            }
                        }

                        //compare words

                        nextWord = false;

                        foreach (string secWord in secWordMean)  //checking if basic word is in
synonym or meaning list
                        {
```

```
                        if (words[i].Equals(secWord) && !nextWord)
                        {
                            //successful match of first word to either meaning or syn of second word

                            words[i] = secWord;
                            freq[i] = freq[i] + freq[j];
                            words.RemoveAt(j);
                            freq.RemoveAt(j);
                            j--;

                            //Console.WriteLine("Successful Synonym Match 1: " + words[i] + " " +
words[j] + " from " + secWord);
                            nextWord = true;
                        }
                    }

                    foreach (string firWord in firstWordMean)
                    {
                        if (words[j].Equals(firWord) && !nextWord)
                        {
                            //successful match of second wordd to either meaning or syn of first word

                            words[i] = firWord;
                            freq[i] = freq[i] + freq[j];
                            words.RemoveAt(j);
                            freq.RemoveAt(j);
                            j--;

                            //Console.WriteLine("Successful Synonym Match 2: " + words[i] + " " +
words[j] + " from " + firWord);
                            nextWord = true;
                        }
                    }

                    foreach (string firWord in firstWordMean)
                    {
                        foreach (string secWord in secWordMean)
                        {
                            if (firWord.Equals(secWord) && !nextWord)
                            {
                                //successful match of one of the first words meanings or synonyms to
either the meaning or syn of second word

                                words[i] = secWord;
                                freq[i] = freq[i] + freq[j];
                                words.RemoveAt(j);
                                freq.RemoveAt(j);
                                j--;
                                //Console.WriteLine("Successful Synonym Match 3: " + words[i] + " " +
words[j] + " from " + firWord + " to " + secWord);
                                nextWord = true;
                              }
                            }
                          }
                        }
                      }
                    }

                    //report on crawling results

                    for (int i = 0; i < words.Count; i++)
                    {
                       writer.Write(words[i] + "\\" + freq[i] + ", ");
                    }
                    writer.WriteLine("\r\n\r\n");

                  }
                  writer.Close();

                  Console.WriteLine("Cralwer has finished.  Resuming normal logging");
                }
              }
            }
```