TRACT: Threat Rating and Assessment Collaboration Tool[1]

A Major Qualifying Project Report:

Submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

in partial fulfillment of the requirements for the

Degree of Bachelor of Science

By

____

Robert Hollinger

____

Doran Smestad

Date: October 17, 2013

MIT Lincoln Laboratory Advisor: Philip Marquardt

____

Professor George Heineman, Major Advisor

[1] This work is sponsored by the Assistant Secretary of Defense for Research & Engineering under Air Force Contract #FA8721-05-C-0002. Opinions, interpretations, conclusions and recommendations are those of the author and are not necessarily endorsed by the United States Government.

Table of Contents

Table of Figures

Abstract

Over a nine week time period we designed and developed a prototype threat rating and collaborative assessment tool for MIT Lincoln Laboratory. This tool is designed to help network analysts identify and monitor emerging vulnerabilities in computing software and determine the level of threat discovered vulnerabilities poses to Lincoln Laboratory's infrastructure. The developed tool was designed both to be modular and extensible, to allow for further development and advancement of this technology at Lincoln Laboratory, and to facilitate teamwork between otherwise isolated analysts as they work to identify threats.

Acknowledgements

Philip Marquardt *(MIT/LL)* – Project advisor and sponsor. Lead of LRNOC, spent many hours with the project team designing project specifications and provided any resource the team needed.

David O'Gwynn *(MIT/LL)* – LLCySA Technical Staff, provided knowledge for LLCySA queries and technical advice during the design of our system.

Professor George Heineman *(WPI)* – Advisor to the project, provided advice on the software modularity, and best design practices.

Executive Summary

MIT Lincoln Laboratory, a federally funded research and development center working under the purview of Massachusetts Institute of Technology and the Department of Defense, is responsible for the creation of advanced technologies in support of national security (MITLL: History, 2013). Lincoln Laboratory's Information Services Department (ISD) ensures the operational integrity of the Laboratory's information technology infrastructure (MITLL: Services Department, 2013). ISD analysts constantly monitor online sources to remain up-to-date on the latest information security threats, also known as *cyber threats*. Based upon information gathered, these analysts evaluate the risk that each threat poses to the network and take the appropriate measures to ensure the network's security. Lincoln Laboratory's Group 51 conducts research and development in the field of Cyber Systems and Operations, building tools to provide analysts with insight into the current state of the network.

This project works with Group 51 to automate the threat detection process by automatically collecting cyber threat information from online sources and providing this data, with threat assessment information, to analysts. Automating this process will expedite the analyst's threat assessment process allowing them to better ensure the security of the network. The development of this tool will be guided by a set of core goals. First, the project will deliver a working prototype that successfully automates the aggregation of posts from various online sources as well as aiding analysts in the identification and assessment of cyber-threats. Second, the prototype will be designed and developed to be modular and extensible allowing for future expansion. Third, all code will be well documented ensuring that the objectives and function of

each section of code is clear to future developers. Last, the system will incorporate a full set of test cases to ensure the proper functioning of the system.

One of the challenges of developing this prototype is the unique infrastructure at Lincoln Laboratory that the system was built upon. The main part of the application, including the user interface and long-term storage engine, run inside of the Lincoln Research and Network Operations Center (LRNOC). Due to security restrictions systems present within the LRNOC are isolated from the outside world making it impossible to gather information from online sources directly. For this reason, the system is divided into two halves: an **Ingester** that operates outside of LRNOC, collecting data and sending it into the LRNOC and the **DBTransfer** which collects the data for permanent storage and querying by analysts.

Analysts using this application are able to create, run and save searches they conduct over the information held within the system's data store. It is important that searches can be saved for future use, because they are often difficult to write and are typically executed multiple times. Analysts are able to learn from others how to write searches and which search criteria are useful to run. This social media aspect of the system will allow for greater collaboration and productivity among the analysts. Interviews were conducted with analysts at Lincoln Laboratory to guide the process of development and to aid in the brainstorming phase of the system's development. Such interviews supported the central idea that analysts currently use online feeds, such as RSS and Twitter, to monitor breaking news on vulnerabilities. It was also confirmed that this process is time consuming because analysts must inspect vast amounts of useless data before finding specific information related to cyber security or vulnerabilities. Our prototype will simplify this process and saves analysts time, further ensuring the security of the network.

We designed the system to both satisfy the requirements of a proof-of-concept while also aiming to be a fully functional prototype. We held interviews with analysts studying and securing the Laboratory network, and analysts within Group 51 who hold experience with creating cyber visualizations. We were able to ascertain a few data sources analysts would use to stay up to date on the latest security threats; these data feeds range from official origins, such as Microsoft Security blogs, to feeds on Twitter by security enthusiasts, such as Bruce Schneier (https://twitter.com/schneierblog). The team decided to build data handlers for two types of online data: RSS/ATOM syndication format and Twitter. Once we collect the data from specific sources supporting these types (an RSS feed, or Twitter list) we send it into the LRNOC using specialized transfer techniques for further handling.

The team researched the best ways to retrieve information from the vast collection of posts we began to accumulate over the course of our project. After holding a conversation with a member from Group 52 (the Human Language Technology group), and performing literary research we discovered options that would increase the fidelity of our search results but were too complex for the timeline of our project. We decided to implement Regular Expression search terminology to achieve better results compared to simple keyword matching while leveraging the existing functionalities already in existence within MySQL Server and the LRNOC. We designed modularity into our system to support any database system and any form of Information Retrieval for future projects to implement.

The most important aspect of our project is the work we performed to encourage and enable communication and collaboration between analysts engaging with our data. We display to all users the searches frequently run, those searches users have saved, and the refinement process others followed to get to a strong regular expression query. We also highlight the actions other users are currently doing in the system to show other users information that they might have not noticed buried in the data. The results of how successfully users can integrate with the system and the time-savings it causes by ending the need to crawl through the online resources manually are important to our system's success. However, what is vital to how we measure success is whether or not analysts will be able to use this system to effectively collaborate and identify threats in a seamless and efficient manner.

After the completion of our prototype it was important to look over the successes and the shortcomings of what we completed. We reviewed our initial goals and evaluated how the system accomplished those goals. One of our central goals was to create a modular design that would allow for simple and efficient development upon our foundation by laboratory personnel. The design of our system is governed by several interfaces that outline the functionalities of each component of our system. These interfaces allow for individual parts of the prototype to be replaced without requiring modification to the rest of the prototype. Thorough testing was accomplished through JUnit testing and coverage was measured through EclEmma (EclEmma Code Coverage, 2013) (JUnit Testing Framework, 2013). Our prototype has 85% coverage ensuring that all classes created are performing as intended and that the prototype as a whole works together as an integrated system. In addition to this testing, documentation was implemented through JavaDoc, inline documentation, a README file, and a

User’s Manual which together outlines how our prototype is currently functioning as well as an explanation of how analysts should use the system.

Through the use of the Twitter4J library and the Rome library we fully accomplished the goal of supporting Twitter, RSS and Atom feeds (ROME Java Library, 2013) (Twitter4J Java Library, 2013). The chosen storage mechanism of all information gathered from these sources was a MySQL database that supports multiple users querying the database for information simultaneously. This infrastructure was the preferred choice as it eliminated the need to develop a network protocol between a server and the users. Information retrieval, being completed through regular expression matching, was also implemented through the MySQL built-in regular expression query.

The User Interface was implemented using the Swing library build into the Java programming language. Our user interface is centered on providing analysts the ability to run regular expression queries over a set time period. Additional abilities include saving regular expressions, flagging posts, and viewing a “Dashboard” or a quick overview of what others users are doing and what is currently trending in the newest, incoming data. This functionality will be especially useful for analysts looking to monitor timely vulnerabilities as soon as possible so they can ensure their network is protected before the vulnerability ever reaches their network.

Over the nine working weeks at MIT Lincoln Laboratory the team leveraged the resources of Group 51 and ISD to accomplish the goal of creating an automated cyber threat aggregation and evaluation tool for analysts. While the project was scoped down from a fully automated threat detection system to a tool that give analysts the ability to detect these threats in real time, the initial goal of giving analysts the ability to more efficiently search over online content for relevant and important information was preserved.

The delivered prototype is a fully functional system that analysts can use; though there are areas of our system that could still be improved given more time and resources. In our prototype, information retrieval was implemented through regular expression matching; we feel that given greater time and resources more advanced algorithms such as machine learning or Term Frequency – Inverse Document Frequency could be implemented and further the success of this system. We also recommend the continued addition of online cyber security sources to the system; with more of these sources the system will become more robust and useful to analysts.

1 Introduction

MIT Lincoln Laboratory is a federally funded research and development center (FFRDC) working to create advanced technologies in support of national security. In pursuit of this goal, Lincoln Laboratory's Information Services Department (ISD) maintains the Laboratory's information technology infrastructure and controls computer network access. ISD protects the network against the latest emerging information security threats (also called *cyber threats*) by collecting timely and accurate information from numerous sources then implementing the appropriate defenses. ISD also makes use of technologies developed by Group 51, the Cyber Systems and Operations group, that provide insight into the current state of the network (situational awareness) and disseminate commands across the network (command and control). This project automates the collection of threat information from a wide range of online sources then enables analysts to work together and quickly search over and view information that they consider relevant and important.

1.1 Problem Statement

Network security analysts monitor the corporate network and watch for possible security breaches. These analysts maintain a mental model of their network's operational patterns and investigate events that deviate from expected behavior. Additionally, network analysts may watch external security news feeds concerning the latest security vulnerabilities and reported attacks; the analyst then would use this knowledge to put forward the appropriate protections to mitigate the threats. At MIT Lincoln Laboratory, ISD analysts undertake this job and provide recommendations for an appropriate resolution, whether that is an immediate patch, a change in firewall rules, or other more sophisticated options.

These analysts observe a number of external sources, including social media sources, to learn of emerging threats. Analysts must then decide how “threatening” the cyber threat is to Lincoln Laboratory’s network. This process, which will be referred to as the “Threat Assessment Process,” is a manual process that consumes much of the analysts’ time; therefore the goal of this Major Qualifying Project (MQP) will be to develop a system, using the resources of Group 51, to assist in this task. The system will automatically retrieve new content, save the content into a database ready for searching, and allow analysts to search the collected information to identify the new trends in security threats. Having this single repository of searchable data reduces or eliminates the need for analysts to manually review numerous separate feeds of information on the Internet.

1.2 Project Goals

The MQP project has four primary goals. First, the project will develop a working prototype to automate the initial phases of the threat assessment process performed by network analysts. This project will deal with the completely automate the collection phase of an analyst’s threat detection process, as well providing analysts with a means to search over the collected information and see only what is relevant. The system will also help analysts learn about how this threat relates to Lincoln Laboratory’s network but the choice and execution of any mitigation strategies are out of scope of this project. Second, the prototype will be extensible and modular to support future changes and facilitate the integration of more advanced threat detection algorithms. Third, all code written will be thoroughly documented using inline comments, JavaDoc, and technical documentation. Fourth, the system will incorporate automated test cases to validate the proper functionality of the prototype. Fulfilling these goals will ensure that the project delivers a fully working and tested prototype that analysts may use to learn more about potential threats to the security of their network while also providing a foundation for future expansion and research.

2 Background

MIT Lincoln Laboratory operates underneath the purview of Massachusetts Institute of Technology (MIT) and the Department of Defense (DoD). Founded in 1951 to design an air defense system for the United States, the Laboratory has operated for over sixty years developing and rapidly prototyping new technologies in support of national security (MITLL: History, 2013). Despite its deep roots in radar technologies, the Laboratory has grown into many diverse fields of work ranging from the original air and missile defense systems to now include topics such as microelectronic fabrication and securing computer infrastructure (MITLL: History, 2013), (MITLL: Organization, 2013). To accomplish these goals Lincoln Laboratory now employs over 1,700 technical staff and 2,000 subcontractors across eight divisions and forty-nine groups (MITLL: Employment, 2013).

Lincoln Laboratory's research and development efforts are organized into eight distinct divisions, each with a broad focus area and commonly referred to by their numerical title instead of their full name (for example "Division 4" not "Homeland Protection and Air Traffic Control"). Each division is broken into a number of groups having a particular portion of the division's focus area. Division 5's focus area is in Cyber Security and Information Sciences and has the responsibility to create technologies that provide mission support through information derived from the cyber domain. The five groups within Division 5 cover topics such as penetration testing, visualization of network activities, and machine processing of human language. These groups aim to build proof-of-concept prototypes in alignment with sponsor needs and transfer the technology to production environments. This process involves researching the problem, developing the prototype systems and components, evaluating the prototypes, and deploying the finished products. Alternatively, the groups may hand off the completed research to commercial entities for further use (MITLL: Cyber Security and Informational Sciences, 2013).

Group 51 focuses on Cyber Systems and Operations. This involves providing operations mission critical cyber information and sensing layers. Group 51 accomplishes this through the research and development of systems that provide analysts with “cyber situational awareness and command and control in the cyber domain” (MITLL: Cyber Systems and Operations, 2013) Cyber situational awareness, similar to the situational awareness of a physical battlefield, includes monitoring computer networks and infrastructure to evaluate its operational integrity. Situational awareness solutions try to answer questions such as: Is our network under attack? Is there strange behavior on the network that could indicate an intrusion? What are our network’s normal operation patterns and have they changed recently? Being able to provide answers to these questions in real time can prevent mission critical devices from becoming compromised as well as ensuring the security of confidential information.

Group 51 is also responsible for the operation and development of the Lincoln Research Network Operations Center (LRNOC). The LRNOC was created to provide cyber analysis of Lincoln Laboratory’s own network traffic, information technology (IT) logs, configuration data, and security system alerts (MITLL: Cyber Security and Information Science Accomplishments, 2013). The LRNOC also functions as an environment for evaluation and refinement of new techniques and technologies before fully deploying them on an active network. Due to the nature of the information stored in LRNOC, it is located on a network separate from Lincoln's local area network (LLAN.) This network will only allow information to travel across the network into the LRNOC; no network traffic is allowed to leave the LRNOC.

2.1 Network Analysts

Network analysts are responsible for monitoring the computer networks and responding to any detected threats or anomalous traffic that occur. As new threats are discovered, analysts react to prepare their systems to detect and handle attempted exploits before a vulnerable network-enabled device, or host, is compromised. Analysts complete the task of assessing potential threats and how the Laboratory should react by browsing various online resources and gathering information on vulnerabilities which have already been discovered by others; analysts then assess the vulnerability and its potential impact before determining how they should best prepare for it. We term this process the "Threat Assessment Process". Consider the following scenario:

> *Scenario: An analyst monitoring twitter feeds, reading security blogs, and watching mailing lists notices a flurry of activity regarding an Internet Explorer zero-day vulnerability being exploited in the wild. The analyst quickly starts to perform research: how many machines on the network are running IE version 8.72? Of those machines, which have already been compromised? The analyst gathers the information, writes manual queries, and sees 34% of the Laboratory computers are vulnerable yet none show signs of compromise. The analyst then acts to quickly patch those machines and/or firewall them appropriately to prevent exploitation.*

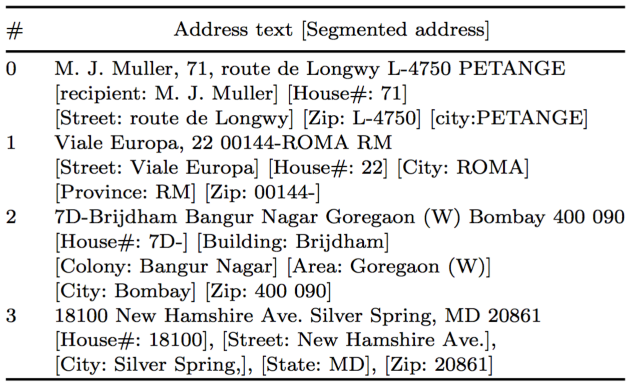
This process takes time because analysts must (1) manually search through numerous online sources for information; and (2) upon discovery of a potential vulnerability, analysts must manually calculate the risk it presents to their organization. In addition, the analyst may underestimate or overestimate the threat of the vulnerability. Lincoln Laboratory's IT department, more commonly known as the Information Services Department (ISD), has a host of analysts that watch the Laboratory network and monitor external sources of information for possible emerging threats against the laboratory (MITLL: Services Department, 2013).

Our project will automate and simplify this process for an analyst. The system developed will automatically gather data (posts) from a list of sources and store them for future use. Gathering such information can be done in two formats: one, actively going out and gathering content ("pulling") or two, having sites send the information directly to us ("pushing.") Pulling would support any website, because a software tool can simply load the page and manually extract the desired content. Pushing would be an efficient implementation, because the software tool would not have to go find the information; it would be automatically sent to the tool by the website as necessary. Once all information is gathered, analysts should be able to quickly search and monitor the discussion of certain topics. For example, an analyst could create a search for "Flash vulnerabilities" and see only the posts that pertain to their topic. This searching process must implement an Information Retrieval technology to determine which posts pertain to a search executed by a user.

2.2 Information Retrieval

Information retrieval (IR) is the process of finding relevant information among a body of work based upon some supplied query. IR technologies generally work to find documents from a given set that pertain to a given topic. The topic itself would be defined by the information retrieval query that is executed (Greengrass, 2013, p. 8). IR technologies vary greatly based upon their implementation and desired results. The most commonly known example of an information retrieval system is a web search engine, such as Google. While IR algorithms used by Google are very sophisticated, more simplistic means of implementing IR could take the form of a Boolean search function that iterates over a collection of words and shows the exact matches it finds (Greengrass, 2013). A more advanced method to perform this task is called *term frequency - inverse document frequency* (TF-IDF), which associates a document with the frequency certain words are repeated in the document; by doing so the TF-IDF has a set of high frequency words which then can be queried and results are ordered by determined relevance (Greengrass, 2013, pp. 19-20). The ordered results derived from TF-IDF is more powerful and useful to an end user, as the Boolean search function simply returns the results that satisfied the query, the results cannot be organized by how well the matched the query. An interview with a member of Group 52 specializing in natural language processing revealed another method of information retrieval, multi-polynomial probabilities. Multi polynomial probabilities are used to make a statement similar to "it is highly probable that the query entered was used to generate this document" and provide a probability-ranked search result; this method, instead of categorizing a search as it applies to documents, can be thought of as evaluating a document for how it applies to a search.

Similarly to IR technology another technology used to find relevant information is called Information Extraction (IE). IE systems, instead of performing a search for key terms provided in a query, process the data and returns meaningful pieces of information (Greengrass, 2013). Sarawagi's information extraction system (Figure 1) is an example in which the system processes a document and pulls out the address information (Sunita, 2008, p. 271). Information extraction can also pull upon multiple documents to gather one piece of information, as it is not pulling pieces of text from the document, but rather developing conclusions based upon all of the information processed (Greengrass, 2013).



#	Address text [Segmented address]
0	M. J. Muller, 71, route de Longwy L-4750 PETANGE [recipient: M. J. Muller] [House#: 71] [Street: route de Longwy] [Zip: L-4750] [city:PETANGE]
1	Viale Europa, 22 00144-ROMA RM [Street: Viale Europa] [House#: 22] [City: ROMA] [Province: RM] [Zip: 00144-]
2	7D-Brijdham Bangur Nagar Goregaon (W) Bombay 400 090 [House#: 7D-] [Building: Brijdham] [Colony: Bangur Nagar] [Area: Goregaon (W)] [City: Bombay] [Zip: 400 090]
3	18100 New Hamshire Ave. Silver Spring, MD 20861 [House#: 18100], [Street: New Hamshire Ave.], [City: Silver Spring,], [State: MD], [Zip: 20861]

Figure 1: Example of the results from an Information Extraction System

Unprocessed date is typically put into one of three categories: a structured source (database), a semi-structured source (webpage), or an unstructured source (raw text) (Chang, Kayed, Girgis, & Shaalan, 2006, p. 1). Information extraction systems typically rely upon preprocessors to use natural language processing for identification of sentence structures and locating parts of speech within the sentence. The extractor is then able to take this identification and process it in a meaningful way.

3 Methodology

3.1 Initial Design

The project started with a meeting between the team and Lincoln Laboratory staff to discuss the goals and expected final deliverable. At first, we were given the project to automatically determine the threat that emerging cyber threats pose to the Laboratory's network. These threats are often reported by various Internet sources ranging from official vulnerability reports to a warning tweet posted by a computer security enthusiast. To determine the specific threat, if any, of these reported cyber threats, the developed system would build a query via information retrieval and information extraction procedures. This query would be executed in order to evaluate threat with relation to the network statistics collected in the Lincoln Research Network Operations Center (LRNOC). For example, if many machines in Lincoln Laboratory were running particular version of a web browser determined to be vulnerable, the threat would be considered high; on the other hand, if very few machines were running the vulnerable software, there would be little threat against Lincoln Laboratory. Later, the project was refocused into a tool that collects information from these Internet sources and compiles them into a searchable structure with a graphical user interface an analyst could leverage to determine the threat level. For the purpose of clarity, this paper will only be discussing our actions in regards to the refocused project.

After this meeting, the first week of the project was devoted to initial designs as well as researching the background knowledge required to proceed. The primary challenge was to deliver a working prototype by October 4th, approximately seven weeks from start to finish. To meet this goal, we chose to use the Java programming language due to its compatibility across operating systems and the pre-existing knowledge held by the team. Java also gives analysts the flexibility to use their preferred system to run our code and greatly simplifies server-side portability. Furthermore, our need for rapid development would be supported by the availability of pre-existing Java libraries that we could leverage to lay the foundation of our program.

Next, we studied the LRNOC's isolated network and the Cyber Situational Awareness (LLCySA) APIs to understand the system's data handling and restrictions to accessing the data. Due to security concerns, the LRNOC is isolated from the outside world by a single one-way link; data to be stored and worked with may only flow into the LRNOC but never out. Therefore, for our system to both receive updates from the Internet and utilize the resources within LRNOC, multiple standalone applications needed to be developed. That is, at least one application lives outside of the LRNOC, passing updates in via the one-way link, and another application inside the LRNOC handles the incoming data. While actually connecting with the LRNOC's API and executing queries is out of the scope of our project, we wanted to have a base level of understanding to construct our system to allow for the addition of direct connectivity to these resources at a later time.

To overcome the limitations of the one-way connection, we established the paradigm of running three separate applications called the **Ingester**, the **DBTransfer**, and the **Application**. At a high level the three-part system was designed to work as follows: The **Ingester** resides on a LLAN (Lincoln local area network) facing server to pull down data from the Internet. It then sends this collected data across the LRNOC's one-way connection to the **DBTransfer**. The **DBTransfer**, running on a server in the LRNOC, reads the incoming data and puts it into a long-term storage repository. Then the **Application** is run on an analyst's machine; functioning as a client, it connects to the long-term storage repository. The Application allows the user to execute search queries over the data. The system is designed to operate with a single instance of the **Ingester** and **DBTransfer** running, but any number of users can be independently running the **Application**.

3.3 Data Collection and Search Algorithms

For our system to be considered useful by an analyst we needed to ensure that the data we collected from the Internet was actually data an analyst would want. To determine what data was useful to analysts we scheduled an interview on August 27th with a network security analyst of Lincoln Laboratory. While we did not discuss any specifics regarding what the analyst does with the information due to operational security, we did learn about a number of the security feeds they monitor. In addition to the normal vulnerability feeds you would expect ISD employs a few authentication-restricted feeds and, interestingly, a number of Twitter feeds. For example, the analysts follow a list of over a hundred Twitter names ranging from the official Microsoft security account (@msftsecurity), to security professionals (Bruce Schneier), to everyday users simply interested in cyber security.

Based upon the information we learned in the interview the team decided to implement two types of data collectors: one for handling RSS/Atom and the other for querying Twitter. We chose only these two as they seemed to be the most relevant to analysts. Other feed possibilities include: Facebook, Reddit, email mailing lists, and a number of more specific pay-to-use sources. The team ruled out support for these (and other) types of data due to both the brevity of the project timeline, and the fact that we are simply building a prototype. Additional sources may be added at any time with minor code changes to our prototype if desired (see Section 4.3 on Data Collection.) After deciding on the collectors we needed to specify the sources those data collectors would process. That is, while we have a collector written for Twitter, we need to specify the Twitter accounts or Twitter searches to retrieve the results from. The team seeded the system with a few RSS feeds we found by quick Google searches for "cyber security RSS feeds," and some sources we learned about from our interviews with analysts.

While this is a naïve approach to selecting strong and reputable sources, the team decided it was acceptable for our prototype. It is intended that analysts input their own feeds they know to be useful. To facilitate this, and forms of more advanced configuration, we designed a command-line interface to manage the sources which the Integer processes the data from, and a properties file allowing more fine-grain control over aspects of the system than just the standard graphical user-interface alone (see Section 4.3 on Advanced User Configurations.)

As data is collected, the Ingester stores the information and has it transported into the LRNOC. Once there it is copied into the long-term storage solution to be held indefinitely. This long-term storage solution can be any database or storage system capable of complying with the needs of the system as specified in our document (see Section 4.1 on Code Design). It is important to keep this information over the long-term so that trends can be established and shown to analysts and it also allows users to ask questions about the data from months ago. These abilities allow an analyst to gain additional background knowledge and perhaps see patterns which would have otherwise been missed. The team made sure to design our requirements for the long-term storage system so that any data is accessible when needed; this ability is strongly wanted by the project sponsors.

Processing analysts' searches over the vast collection of data takes special software to locate the relevant articles; this class of software is called Information Retrieval, or IR. To select the best option available the team met with a member of Group 52, or the language processing group, and inquired about modern day IR systems. The member from Group 52 gave the team a high level understanding of how IR works and common techniques to implement such functionality. Due to the time restrictions on our project, we opted for finding solution which was already in existence and we could easily add to our prototype. After a conversation with the project sponsors it was decided that a search system based on Regular Expressions would suffice as long as our system came with the option to modularly upgrade to a more sophisticated solution in the future (see subsection of Section 4.7 on Information Retrieval.)

3.3 Application & Social Networking

All the work outlined in this section so far has been building the foundation to power our Graphical User Interface (GUI). This interface will be composed of the various windows that an analyst will interact with to utilize the powerful system underneath. The interface packaged with our prototype is not intended to be extended or modified in the future; it is simply a proof of concept to show what the team believes to be just one use case the underlining architecture can be used for. Most notably, we place emphasis on the ability of analysts to build a sense of community knowledge, collaborative threat discovery, and up-to-date information on how other analysts are using the system.

Currently analysts we interviewed go through their process of threat identification by themselves. There is some collaboration between those physically present in the same office, but unless something particularly noteworthy is discovered, an analyst will work in isolation. The team set out to identify how to construct an interface that allows for analysts to transfer information to one another. We do not attempt to replace email or instant messaging systems as most institutions already have those standard forms of communication in place; instead we wanted to allow analysts to build up communal information relating to the data held within our system. To accomplish this we integrated a number of features allowing analysts to mark the information they find, or the procedures to find that information, as something worth sharing with others. Due to time limitations and little concern from the sponsor with regard to privacy of our system's data (especially while hosted within the LRNOC), we did not implement any access controls over shared data.

To allow for shared data between the analysts, the team implemented the ability to save a search performed by a user. In addition to being convenient for future use, this allows an analyst to build up a list of searches they feel are worth repeating and inspecting the results of. To facilitate the growth of community knowledge we allow any user to see the saved searches of any user. Some searches may not apply to all types of analyst work, but we feel this will allow both for the sharing of interesting searches but also unique ways to look for a particular piece of information. Due to the choice of regular expressions as an information retrieval system, building the perfect query might take some work, and once a user arrives at something they believe is ideal, we want to share that item with all users of the system. Leveraging this information the system can track the number of times a search is run and build up a list of popular searches; searches in this list have a decay over time putting only the most recent, repeatedly searched expression at the top. This allows an analyst to quickly identify what others may have already noticed as a threat to watch for.

Second, the team implemented the concept of “refining” a search. When an analyst starts with a search expression, reviews the results, and then decides that the search would be better in a different form, we believe is an important decision to keep track of. Whether the search was too precise to return many results, too generic to give relevant results, or even was simply mis-formatted, our system tracks this change. We gather together these choices and build a directed weighted graph representing how users navigate through our system. This is then displayed in a panel for future users who perform the same search. For example, if a user were to search for vulnerabilities relating to Java 7, there may be too many results to be very useful; the user would utilize our refine feature to follow the path set by a previous analyst to “Java 7 vulnerabilities” or some other more precise search. The refine options presented to a user are organized by popularity and then decay with time, keeping the list up to date with the recent refinements as search results change over time.

Third, the team wanted to have a notion of posts worth more attention by all analysts. We also wanted this notion to be global, because if one analyst believes it is important; all analysts should be alerted about the post. After an analyst performs a search and reviews the results he or she may come across a post they wish to mark as interesting. This could be for any reason deemed appropriate by the analysts of the system: for example they could agree a flagged post means a vulnerability needing to be looked into. The analyst who reviewed and dealt with the possible threat would then remove the flag indicating the post is no longer a concern.

With all of the data held in the system, the project sponsor asked the team to develop a panel providing an instant overview to the system. We took this idea and expanded upon it to give analysts the ability to get a quick update on the current trends in online sources and updates on user activity within the system. To support this, a “dashboard” was developed that includes a graph of how many posts several generic regular expressions matched over the past day, a feed of the most recent analyst activity, and a feed of the most recent relevant posts added to the system. The graph can easily and quickly show analysts if there was a spike for a regular expression potentially representing a new vulnerability. The feed of user activity can also be used by analysts to learn of the searches other analysts are running and create their own searches accordingly. Lastly, the feed of posts shows the posts most recently added that match one of the regular expressions in the graph. This panel is meant to give context to a spike in the number of posts matching a regular expression. The dashboard can be used by individual analysts to cater their searches according to live data, or it could be displayed all the time on a big screen for constant monitoring. In this case, the application could then be run to gain more information if an analyst found something on the dashboard interesting.

Finally, we designed how our application would interface with the LLCySA APIs to provide an analyst with a step forward after discovering something interesting within our prototype. We aimed to have our system construct the query for the LLCySA system that an analyst could copy, paste, and execute the query. Unfortunately due to technical problems outside of our control, we were unable to fully complete this communication. Instead, we put together a couple of specific use cases and detailed how the LLCySA system could be connected to our system after our project ends. We were able to implement a first pass of generating a syntactically correct query as a proof-of-concept, but were not able to process much past this point. The team left detailed documented instructing how to add this functionality for the next team.

3.4 Prototype Testing and Evaluation

After developing the prototype, we evaluated our prototype and measured how successful the system was in accomplishing its goals. First, it is important to look back upon the core goals defined at the start of our project: a working prototype, a modular design, complete documentation, and a complete suite of test cases. If accomplished, these goals would ensure that the prototype is both functional and easy for Lincoln Laboratory to build upon in the future. It is also important to ensure that the prototype is running efficiently and can expedite the threat detection process for analysts. To measure this we evaluated the information retrieval technology implemented as well as ensuring that our system returned results to analysts in a timely fashion. These statistics were measured and tracked throughout the development process to monitor the progress of development and to ensure that the libraries and algorithms being implemented satisfied the system’s requirements.

4 System Architecture and Design

4.1 Code Design

The overarching goal for all code written was to ensure it was modular in design, easily extended, well documented, and thoroughly tested. All class and interfaces mentioned in this section can be seen with more detail in our UML diagram found in Appendix A. The code base currently holds four fundamental interfaces which, when called in order, process and display the output we intend. First, we have the *IExtract* interface; this interface dictates the functions required for a new source type to be added into the Ingester for retrieving data from the Internet. Classes which implement this interface are expected to make their own web-requests (if appropriate) and return a collection of posts to be handled by the system. Our prototype was delivered with two fully functional classes implementing *IExtract*: a RSS/Atom feed processor, and a Twitter feed processor. Second are the *IStorage* interfaces, these interfaces dictates how code dealing with an underlying data storage mechanisms must interact with our system. There are two interfaces dealing with storage, one for the information being transferred from the Ingester into the LRNOC (*ITransportStorage*), the other handling the long-term storage of data after it arrives in the LRNOC (*IStorage*). The last interface implemented is *IInformationRetrieval*, responsible for all classes that conduct information retrieval on the data stored. The implementation of these interfaces allows future developers to easily replace parts of our prototype with more advanced technologies. For example, if a new information retrieval class was developed it could be ‘plugged into’ our code as long as it follows the rules defined by the *IInformationRetrieval* interface.

4.2 Advanced User Configurations

The team believes that the code fulfills the modularity and extensibility parts of the original goals set, however we also wanted a user to be able to modify the system in custom ways without needing to edit and recompile the code. To accomplish this we developed two methods by which an advanced user can customize our system. First, we built a command-line interface for manipulating the sources pulled by the Ingester. We realized during our own testing that a user may want to add or remove sources at any time while the Ingester is running. Since the sources are held within a SQLite database, manual modification is nearly impossible; therefore the command-line interface we bundled with the Ingester provides the user with the ability to list, add, or remove sources. Each source contains all of the information that the Ingester needs to pull information from that source including a type, an identifier, as well as common name that allows an analyst to quickly identify a source. Second, we implemented a properties file allowing an advanced user more control over some fine points within the system than provided by the graphical user interface. A user can modify this file with any text editor and control various program properties such as the update frequency of the Ingester.

4.3 Data Collection

First, we wanted to be able to process RSS and Atom feeds. RSS (Rich Site Summary) typically is included alongside many blogs or constantly updating sources of information; being a syndication format it is designed to provide information about the website in a particular format. Atom (Atom Syndication Format) feeds may be thought of as a newer revision of RSS and supporting both of these formats allows our system to interface with a large number of websites. Building a system to process the RSS and Atom XML formats is beyond the scope of our project, instead we decided to leverage the Java library ROME which automatically pulls and formats information from all versions of RSS and Atom feeds (ROME Java Library, 2013). Second, following from our discussion with the Lincoln Laboratory analyst, we also needed to have support for processing Twitter feeds. Unfortunately the latest version of the Twitter API no longer supports the generation of RSS feeds; therefore we needed to communicate using the JSON format standards. Instead of manually trying to work with JSON, or even using a JSON library to interact manually with the Twitter API, we utilized the Java library Twitter4J. Twitter4J is the *de facto* standard for interaction between Java and Twitter although Twitter does not officially endorse it (Twitter4J Java Library, 2013). Any number of these data collectors may be added to the system, no limitation has been put into place; for example, with minor code modifications you could support Facebook, Google Plus, or even a proprietary format.

In implementing the collection processes to pull the data from our sources we needed to identify how we would receive updates: either through push notifications, which are more efficient, or through polling of the site, which is simpler. Taking a look at our source types, RSS and Atom feeds make no provision for push messaging as they are simply an XML formatted page expressing the already existing content on a website. The only way to receive an update for an RSS feed is to ask the website if there have been any changes since the last time you asked, if there have been, you download the new updates. Twitter on the other hand definitely supports push messaging for the notification of new tweets. Instead of continuously asking for update information Twitter is able to simply send you a message when an update arrives so that you can retrieve it. Yet, since RSS feeds do not support push messaging, the team decided to simply use polling for both RSS and Twitter data to keep the system design simple. Since our system must make the request in a polling system, a polling interval must be set. After numerous rounds of Ingestion the team believes a ten-minute interval is a fair balance between timely updates and fruitless requests. The polling interval can be changed in the properties file, but we decided upon this default value based upon the frequent updates of twitter feeds, and trying to keep a minimum delay between a post going live and an analyst able to query the post. When the polling interval elapses, our system pulls all the data it can find and creates a collection of post objects. We expect the system to be able to handle many hundreds if not thousands of posts per polling cycle. These post objects might have duplicate entries compared to the last time we pulled, therefore our storage mechanisms is designed to properly handle duplicate data.

4.4 Gathering Posts

Once the Ingester receives all the data, it needs to be put into a standardized format that our system can handle. The team decided to call this structure a “post.” A single post holds information that is expressed in a single tweet, or a single RSS entry. A post is an object holding Source Name, Title, Date, Author, and the body of the posted message (tweet / RSS entry). If a round of Ingestion pulls 234 tweets from Twitter and 23 entries from an RSS feed, there will be 257 post objects created. We decided to use a SQLite database to transfer these post objects from the Ingester to the DBTransfer. We felt that this was a simpler solution as it avoided the need to serialize the object for storage in a text file or other custom data type; additionally since no encoding is necessary a SQLite browser is able to quickly display the contents of the

Post table, this is very helpful when debugging the system. When the DBTransfer receives the SQLite database it transfers the posts into a MySQL database for permanent storage.

4.5 MySQL Database

During the design of the three separate parts of our threat aggregation and assessment system we realized that to have reliable and stable storage we would need a strong database backend. Additionally, we would need to support a number of simultaneous connections to serve multiple analysts querying the system at the same time. Database servers are specifically designed to address these needs. After a discussion with the project supervisors, it was decided that we would use a SQL-based storage mechanism instead of a No-SQL implementation due to the restricted time frame of the project and pre-existing knowledge held by the team. The team chose to use a relational database server due to the connections and interweaving of information our tool requires: for example, we tie a username to the searches the user has performed. After a quick review of the options available to us, we decided to use MySQL Server. MySQL Server is a production-grade open-sourced database management system maintained by Oracle and is designed to handle heavy use in an efficient and quick manner. The choice to use MySQL Server over other offerings such as Oracle, Postgres, or MariaDB, was simply a matter of convenience and the use of a well-known and trusted product.

Following closing with proper normalization techniques we designed a number of tables to support the queries our system performs. A full description of the tables and how they are able to interact may be found in Appendix A. Each table has an explicitly defined primary key. Most of our tables use the standard auto-incremented integer for row identification; however, for some tables we decided to use an inserted value as the key, such as Dimensions or Source Types. These values are usually very short one or two-word phrases and therefore we felt that simply using them as the primary key was more efficient than doing the extra table lookup to discover which word maps to which automatically incremented integer. We also have defined foreign keys and foreign key constraints to aid in the speed of our table joins and ensure that our data remains internally consistent. That is, upon insertion of a new row any reference to a different table is validated to be sure the row exists, and upon deletion of a row any row depending upon the row deleted is also removed from the database.

The database table Post which is intended to hold all unique posts ever gathered by the Ingester is expected to grow to be quite large. In order to maintain fast query times even when there may be three million posts stored we added an index over the date column. The application always constructs a query restricted to a particular timeframe that should be able to keep regular expression query times to under a second. Similar in fashion to how we handled duplicate posts in the SQLite transport database, we established a unique constraint on the data field for the Post table. Since we implemented the data field as a MEDIUMTEXT data type (capable of holding up to 16MB of information per row), the unique constraint needed to be limited in scope. We decided that forcing the first 512 characters to be unique from row to row was a good approach; it is short enough to facilitate quick insert times, yet long enough to safely keep duplicate entries out of the system. This decision does have some effect on insertion time, as the database must search over the entire table upon each and every insert of a post. While this will cause the inserts to become slower and slower the more posts are added, we decided this would not be a huge concern as the Ingester polling period will certainly be longer than the time it takes to insert the queries; however, it would be our recommendation that if this prototype is intended for production-level use a more sophisticated method of preventing duplicated posts should be implemented.

4.6 Flow of Information

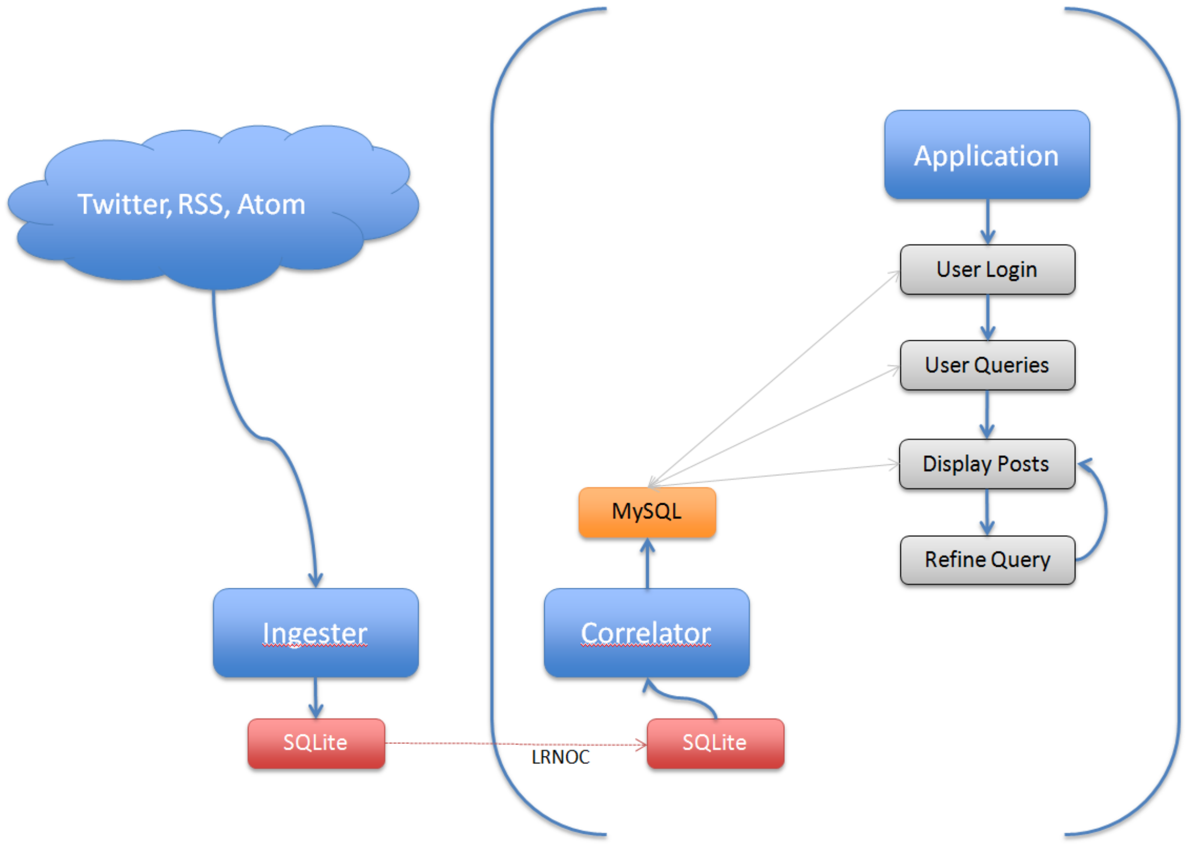


Figure 1: The figure depicts the flow of information through the system beginning with the Ingestion phase and ending in the user interface.

4.7 Sections of Development

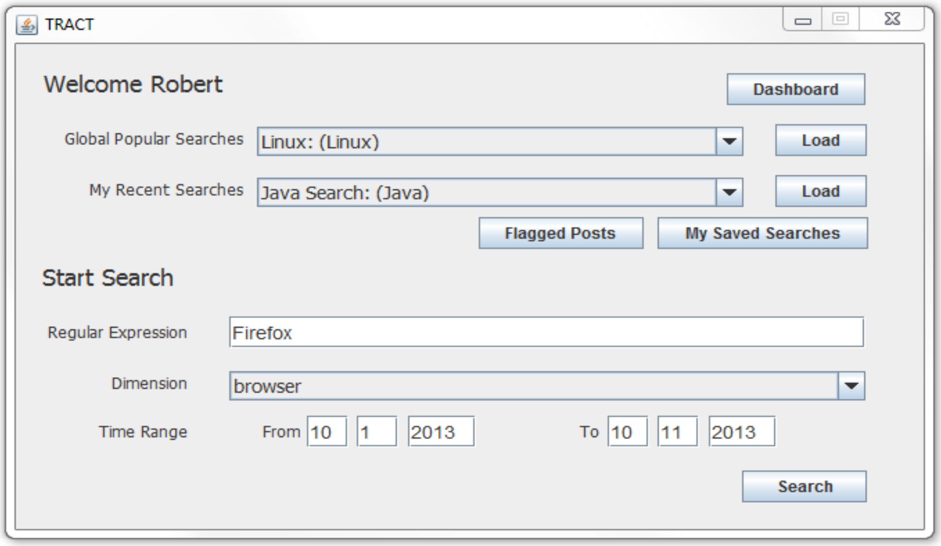
Information Extraction

Once all posts have reached the LRNOC they are stored into a mySQL database for future use. The next phase of our program will need to run some information extraction technologies to determine which posts actually contain information relevant to the needs of an analyst. Information extraction technologies such as text matching, regular expression matching, and more sophisticated tools such as Lucene, a Java based indexing and search technology, were considered. Strict keyword matching was determined to be too confining, as we are reading from user inputted text talking that can say the same thing in many different ways. For example, we want to match 'Firefox' as well as 'firefox' in the same search. Looking further, we also want to match on what version number of Firefox these specific posts are talking about without having to define a keyword for every possible version number. Next regular expression matching was considered as it allows for all of the abilities that keyword matching lacked in a similar fashion. Lucene was also considered as it is a much more advanced search system. After consideration and discussion with the Lincoln Laboratory team, it was determined that regular expression matching would fulfill the requirements of our information extraction algorithm, while maintaining our timeline.

The information retrieval algorithm implemented in our project will be dealing with semi-structured data being pulled from various sources. Our data is semi-structured as it is coming from feeds, governed by a specific format. This format allows us to pull the specific parts of the posts out individually. For example, we can easily get the author, the title, and the date in which it was published without doing any real information extraction. The actual content of the posts can also be pulled out; however this section in itself is unstructured as an author can type anything, in any format in this section. Information retrieval technologies will need to be implemented allowing the system to determine if a post pertains to a query that a user executed.

User Interface

A user interface was developed as a part of the prototype to allow analysts to interact and learn from the data collected in the ingestion phase. The first phase of this application will be a log in screen where the user enters access credentials, or creates an account. All queries executed will be tagged with this user's username. This will allow other users to see what queries were executed and by whom in the future. Next, the user will need to select an already created regular expression query, or create their own query along with a time frame (past day, week,


TRACT

Welcome Robert

Global Popular Searches: Linux: (Linux)

My Recent Searches: Java Search: (Java)

Start Search

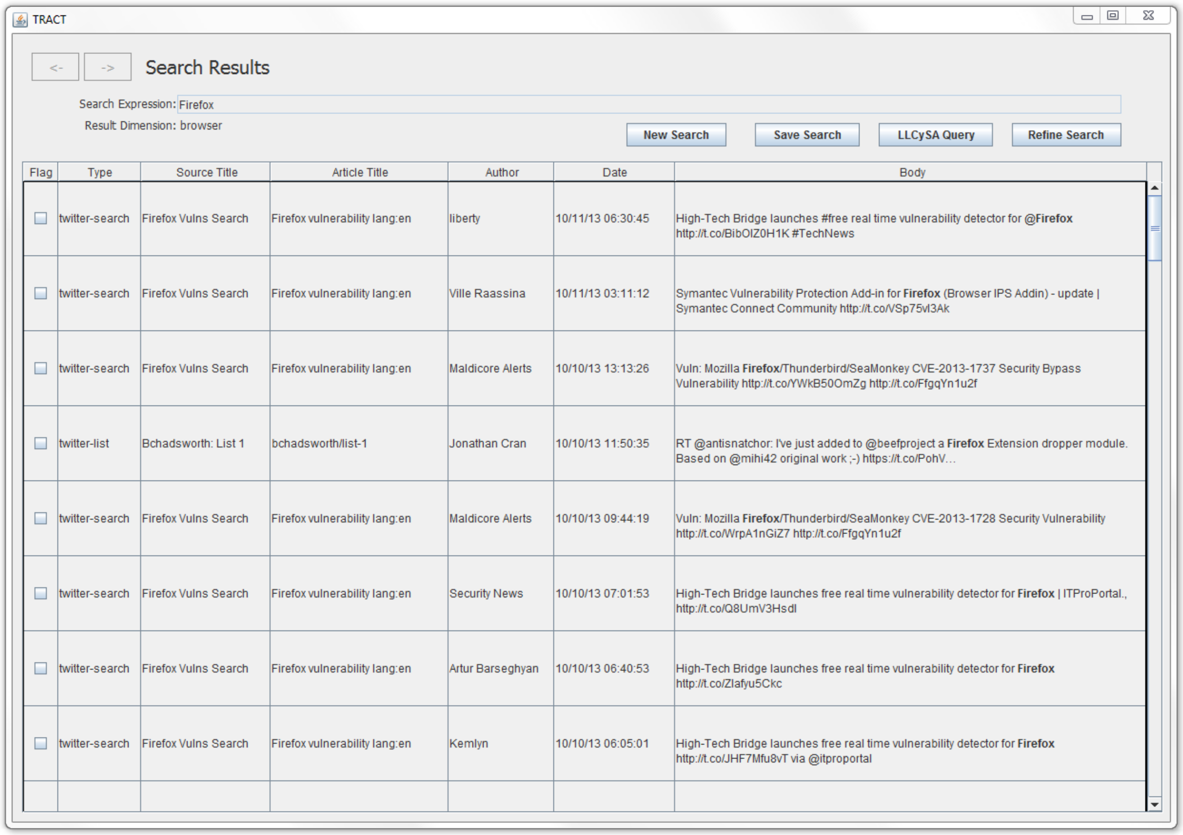
Regular Expression: Firefox

Dimension: browser

Time Range: From 10 1 2013 To 10 11 2013

Figure 2: TRACT Welcome Screen

month, etc.) and run it on the posts collected (See Figure 2). We determined that it would be useful for a user to have the ability to save a search that they conducted for future use. The system will also show each user, a list of the most popular searches, among all users of the system. This functionality will allow users to find valid regular expression that are popular



Search Expression: Firefox
Result Dimension: browser

Flag	Type	Source Title	Article Title	Author	Date	Body
☐	twitter-search	Firefox Vulns Search	Firefox vulnerability lang:en	liberty	10/11/13 06:30:45	High-Tech Bridge launches #free real time vulnerability detector for @**Firefox** http://t.co/BibOIZ0H1K #TechNews
☐	twitter-search	Firefox Vulns Search	Firefox vulnerability lang:en	Ville Raassina	10/11/13 03:11:12	Symantec Vulnerability Protection Add-in for **Firefox** (Browser IPS Addin) - update | Symantec Connect Community http://t.co/VSp75vl3Ak
☐	twitter-search	Firefox Vulns Search	Firefox vulnerability lang:en	Maldicore Alerts	10/10/13 13:13:26	Vuln: Mozilla **Firefox**/Thunderbird/SeaMonkey CVE-2013-1737 Security Bypass Vulnerability http://t.co/YWkB50OmZg http://t.co/FfgqYn1u2f
☐	twitter-list	Bchadsworth: List 1	bchadsworth/list-1	Jonathan Cran	10/10/13 11:50:35	RT @antisnatchor: I've just added to @beefproject a **Firefox** Extension dropper module. Based on @mihi42 original work ;-) https://t.co/PohV…
☐	twitter-search	Firefox Vulns Search	Firefox vulnerability lang:en	Maldicore Alerts	10/10/13 09:44:19	Vuln: Mozilla **Firefox**/Thunderbird/SeaMonkey CVE-2013-1728 Security Vulnerability http://t.co/WrpA1nGiZ7 http://t.co/FfgqYn1u2f
☐	twitter-search	Firefox Vulns Search	Firefox vulnerability lang:en	Security News	10/10/13 07:01:53	High-Tech Bridge launches free real time vulnerability detector for **Firefox** | ITProPortal., http://t.co/Q8UmV3Hsdl
☐	twitter-search	Firefox Vulns Search	Firefox vulnerability lang:en	Artur Barseghyan	10/10/13 06:40:53	High-Tech Bridge launches free real time vulnerability detector for **Firefox** http://t.co/Zlafyu5Ckc
☐	twitter-search	Firefox Vulns Search	Firefox vulnerability lang:en	Kemlyn	10/10/13 06:05:01	High-Tech Bridge launches free real time vulnerability detector for **Firefox** http://t.co/JHF7Mfu8vT via @itproportal

Figure 3: TRACT Search Results

among many users to start their searches off of. Once a search is run a list of posts matching that regular expression will be presented (See Figure 3). The user will have the ability to continually refine their regular expression and see how the list of posts that match the regular expression change with each refinement. When refining a regular expression, the user will be shown a graph of popular refinements based upon how other users refined this exact search in the past as well the refinements that lead other users to this search (See Figure 4).

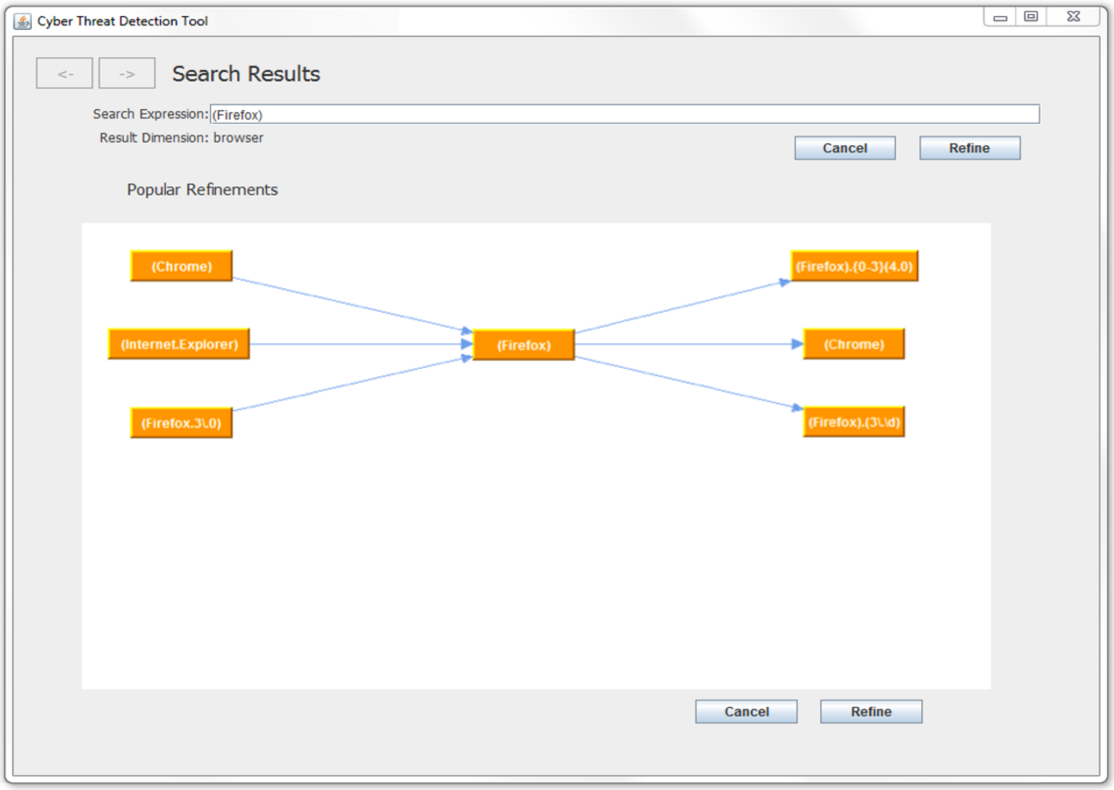
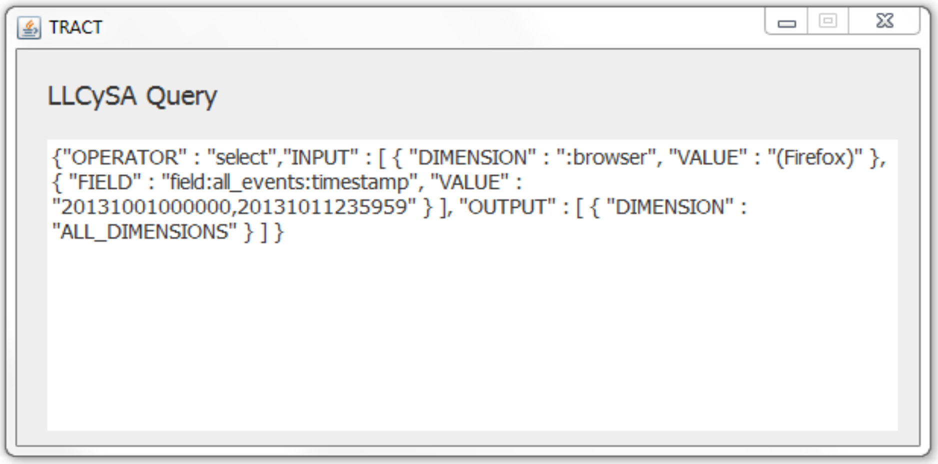


Figure 4: TRACT Refinement Process

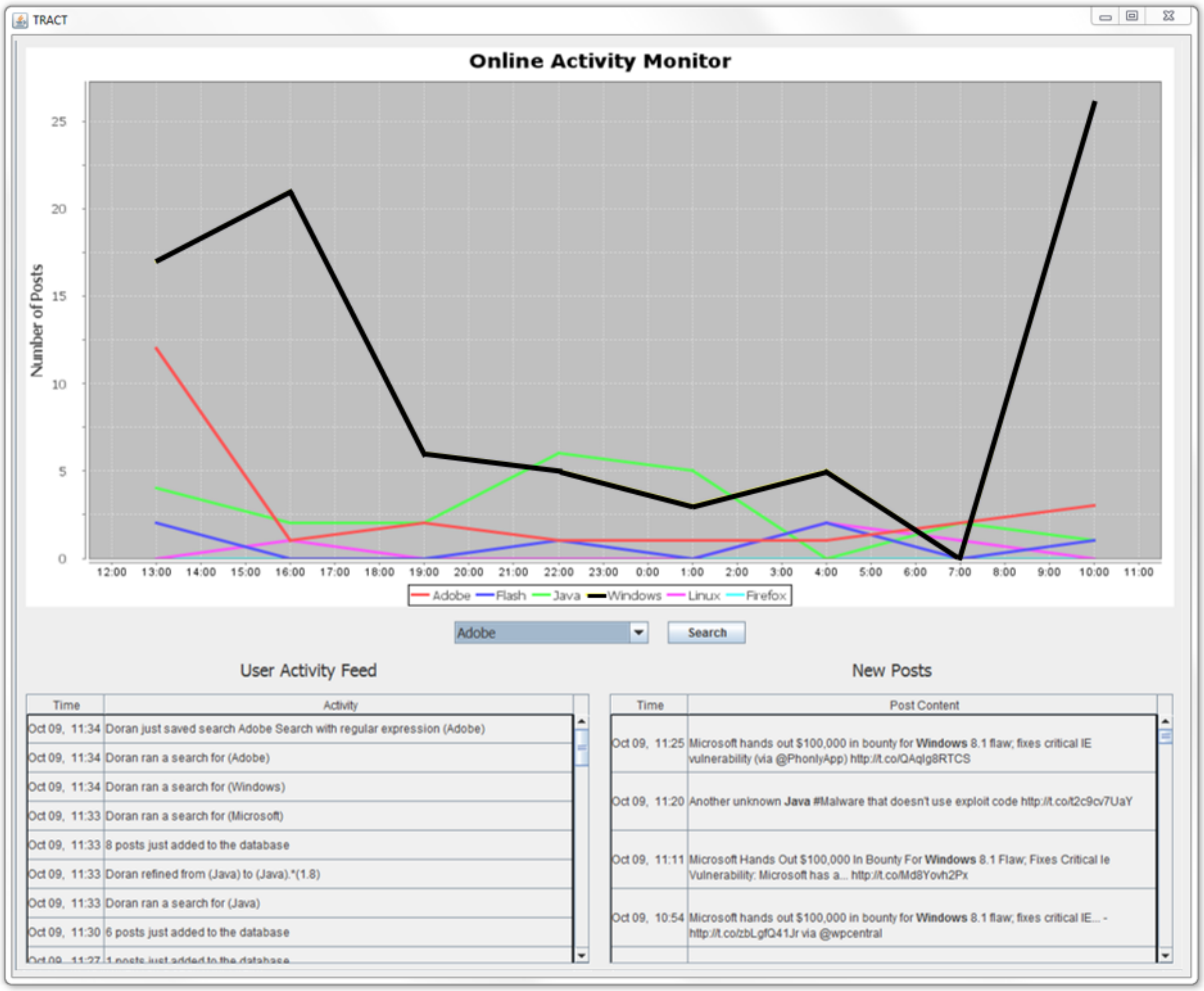
The user will also have a chance to generate an LLCySA query based upon the regular expression search that was conducted and how the posts matched that regular expression (See



```
{"OPERATOR" : "select","INPUT" : [ { "DIMENSION" : ":browser", "VALUE" : "(Firefox)" },
{ "FIELD" : "field:all_events:timestamp", "VALUE" :
"20131001000000,20131011235959" } ], "OUTPUT" : [ { "DIMENSION" :
"ALL_DIMENSIONS" } ] }
```

Figure 5: TRACT LLCySA Query

Figure 5). The information returned from the LLCySA query will allow users to obtain information on how much of threat found vulnerabilities actually pose to Lincoln Laboratory's network. The final piece of the User Interface is the Dashboard. The Dashboard is intended to give users a quick glance at what is happening right now and can quickly display what is 'trending' (See Figure 6.)



User Activity Feed

Time	Activity
Oct 09, 11:34	Doran just saved search Adobe Search with regular expression (Adobe)
Oct 09, 11:34	Doran ran a search for (Adobe)
Oct 09, 11:34	Doran ran a search for (Windows)
Oct 09, 11:33	Doran ran a search for (Microsoft)
Oct 09, 11:33	8 posts just added to the database
Oct 09, 11:33	Doran refined from (Java) to (Java).*(1.8)
Oct 09, 11:33	Doran ran a search for (Java)
Oct 09, 11:30	6 posts just added to the database
Oct 09, 11:27	1 posts just added to the database

New Posts

Time	Post Content
Oct 09, 11:25	Microsoft hands out $100,000 in bounty for **Windows** 8.1 flaw; fixes critical IE vulnerability (via @PhonlyApp) http://t.co/QAqlg8RTCS
Oct 09, 11:20	Another unknown **Java** #Malware that doesn't use exploit code http://t.co/t2c9cv7UaY
Oct 09, 11:11	Microsoft Hands Out $100,000 In Bounty For **Windows** 8.1 Flaw; Fixes Critical Ie Vulnerability: Microsoft has a... http://t.co/Md8Yovh2Px
Oct 09, 10:54	Microsoft hands out $100,000 in bounty for **Windows** 8.1 flaw; fixes critical IE... - http://t.co/zbLgfQ41Jr via @wpcentral

Figure 6: TRACT Dashboard

4.8 LRNOC Interface

The final stage of this prototype is an LRNOC interface. The team implemented a Java Interface object to dictate how classes that construct LLCySA queries should operate; in short construction methods need to provide a string output we are capable of displaying to the end user. Our user interface will assist an analyst form an LLCySA query that pertains to the information they are searching on in our application. For example, if a user is searching for vulnerabilities in Firefox version 17.0.3, then we will construct an LLCySA query to determine how many users are currently running on Firefox version 17.0.3. The results of this operation are a query string, or prepared JSON statement to send towards the appropriate LLCySA query. The query generated is based upon the regular expression, search dimension, and date range supplied by the analyst. When the generated query is run it will allow an analyst to determine how much of a threat such vulnerability actually poses to the Lincoln Laboratory network. If only a few computers on running this version they then threat is minimal, however if half are running it the threat is much more significant.

5 Results and Discussion

After the completion of our prototype it was important to look over the successes and the shortcomings of what we completed. First, we looked over our initial goals and evaluated our system in how well it accomplished those goals.

5.1 Modularity

One of our central goals was to create a modular design that would allow for simple and efficient development upon our foundation by laboratory personnel. The design and back end of our system is governed by several interfaces that outline the functionalities of each component of our system. The Ingester is guided by *IExtract* in such a extension of our system to extract data from currently unsupported source type, such as Facebook or specific websites, is as simple as creating an addition class the implements the *IExtract* interface. This class will pull information from the desired source and store the information in the correct format. This extension of additional sources does not place and limitations on the types of sources that we can pull information from or the way in which the implementation is done.

The information retrieval phase of our system is guided by the *IInformationRetrieval* interface. This interface provides the guidelines for an information retrieval system running on the collected data. The information retrieval system was once again created to be modular and the entire system can easily be changed out for more sophisticated, advanced technologies such as machine learning or TF-IDF. Replacing the current regular expression matching technology with more advanced technologies such as again just requires the creation of a class that implements the *IInformationRetrieval* interface and then swapping out the current regular expression matching technology with this technology in the system.

Lastly, our prototype implemented SQLite and MySQL databases governed by *ITransportStorage* and *IStorage* interfaces. Again, these interfaces ensure a modular design and allow future uses to implement other storage mechanisms with relative ease and minimal changes to our existing system. We feel that through the implementation of these interfaces, the modularity of these components was successfully met. The one component of our design that is not modular by design is the User Interface. Our user interface is specifically created to show the kind of information that our system is currently providing. We are running information retrieval through regular expression matching, so the User Interface is centered on this functionality. If more advanced information retrieval technologies are put into place, we feel that the user interface should be recreated to better suit the needs of the technology implemented.

5.2 Testing and Documentation

A complete set of testing and documentation was a central goal of our project. We implemented a testing suite through JUnit and measured testing coverage through EclEmma (JUnit Testing Framework, 2013) (EclEmma Code Coverage, 2013). Our prototype has 85% coverage. This coverage ensures that our all classes created are performing as intended and that the prototype as a whole is well tested to ensure both each part is able to work together as an integrated, fluid system. Documentation is complete with both technical and in line documentation. All documentation follows JavaDoc standards to ensure that it is easy understood and traversed by future developers. All classes, interfaces, and functions have documentation explaining their function as well as how it is being used to in our system as a whole. In line documentation is used to explain in more detail what the purpose of each section of code and how it is being implemented. Documentation also works as a basic in which future developers can learn how our system functions, the technologies we implemented, and how to extend or replace parts of our code to develop a more advanced prototype. In addition to this documentation a readme file is also included that outlines how our prototype is currently functioning as well as an explanation of how an analysts should use the system.

5.3 Ingester

At the start of our project we set out with the goal of supporting Twitter, RSS and Atom feeds. Through the use of the Twitter4J library and the Rome library we fully accomplished this goal (Twitter4J Java Library, 2013) (ROME Java Library, 2013). One unforeseen effect of using these libraries is that many RSS feeds created are not correctly formatted. The Rome library would often fail to ingest on these sources and would provide inaccurate information leading to our system not getting the data, or crashing due to improperly formatted text. For this reason, we currently support correctly formatted RSS feeds. It is recommended that all RSS and Atom feeds being checked for proper formatting before being added to the system. Failure to do so could cause inaccurate information to be added to the system. RSS feeds can easily be checked for any formatting errors through web services such as feedvalidator.org (Feed Validator, 2013). Another limitation on our system is the number of posts we can pull in at once, for RSS and Atom feeds we are able to pull in the first page of posts, which for most feeds is twenty posts, and we can pull in the last 100 posts from Twitter at once. This limitation means that we have to be constantly pulling from these sources to ensure that we do not miss any posts. In the early stages of our data collection we found that when pulling from five RSS feeds and one twitter list of almost two hundred users, every ten minutes we were getting a total of between five and twenty-five posts per pull. The majority of these posts were coming from the twitter list and each RSS feed was contributing at most five posts in one pull. Therefore, it was concluded that a pulling from each data source every ten minutes prevented us from missing any posts. Ten minutes is also an appropriate amount of time as it allows our application to remain up to date at all times. We recommend that the timing interval between pulling from sources remain at or under ten minutes to ensure that an analyst using the application is getting up to date and accurate information.

5.4 Prototype Backend (Information Retrieval and Database)

The back end framework of our prototype is divided into two central parts, the database governed by the *IStorage* interface, and the information retrieval technology implemented by the *IInformationRetrieval* interface. The storage mechanism chosen was a MySQL database with multiple users querying to that database for information at the same time. This infrastructure was the preferred choice as it cut out the development of a network protocol between a server and the users as we use the database as this server and the users can directly query it. Information retrieval, being completed through regular expression matching, was also implemented through the MySQL built in regular expression query. Once implemented we needed to ensure that this infrastructure would be fast enough to handle all of the posts being added to the system and then a user querying over the entire database for posts matching a regular expression. Testing revealed that after the addition of millions of posts to the database the regular expression testing did being to slow down and complex queries would take over a second to return. This risk was mitigated by configuring the database to automatically sort posts by the date in which they were posted. Each search that a user is executing is over a set time period and we envision that this time period would be over the course of a days or weeks not years as an analyst will want to be getting up to date information. Providing this constraint each search query is only running a regular expression search on the posts that are in the desired time range, which is generally in the range of hundreds or possibly thousands of posts. This constraint noticeably speeds up search queries, into the range of tenths of seconds for complex searches, successfully mitigating the risk.

5.5 Prototype Frontend (User Interface)

The User Interface was implemented through the Java built in Swing library. Our user interface is centered on giving an analyst the ability to run regular expression queries, over a set time period. Additional abilities such as saving regular expressions, flagging posts, and viewing a "Dashboard" or a quick overview of what others users are doing and what content is currently highly active on. Highly active content given an analyst the ability to see what searches may be interesting at the current point in time. For example, a spike in activity on the word "Java" may indicate that there was a recently found vulnerability in Java the many users are currently talking about. We found this tool to be successful in identifying such new vulnerabilities on October 4, shortly after a major breach in Adobe software. Our tool was able to quickly and efficiently show a massive spike in activity on the work “Adobe”, as seen in Figure 7. This functionality will this will be especially useful for analysts looking to catch breaking vulnerabilities as soon as possible so they can ensure their network is protected before the vulnerability ever reaches their network.

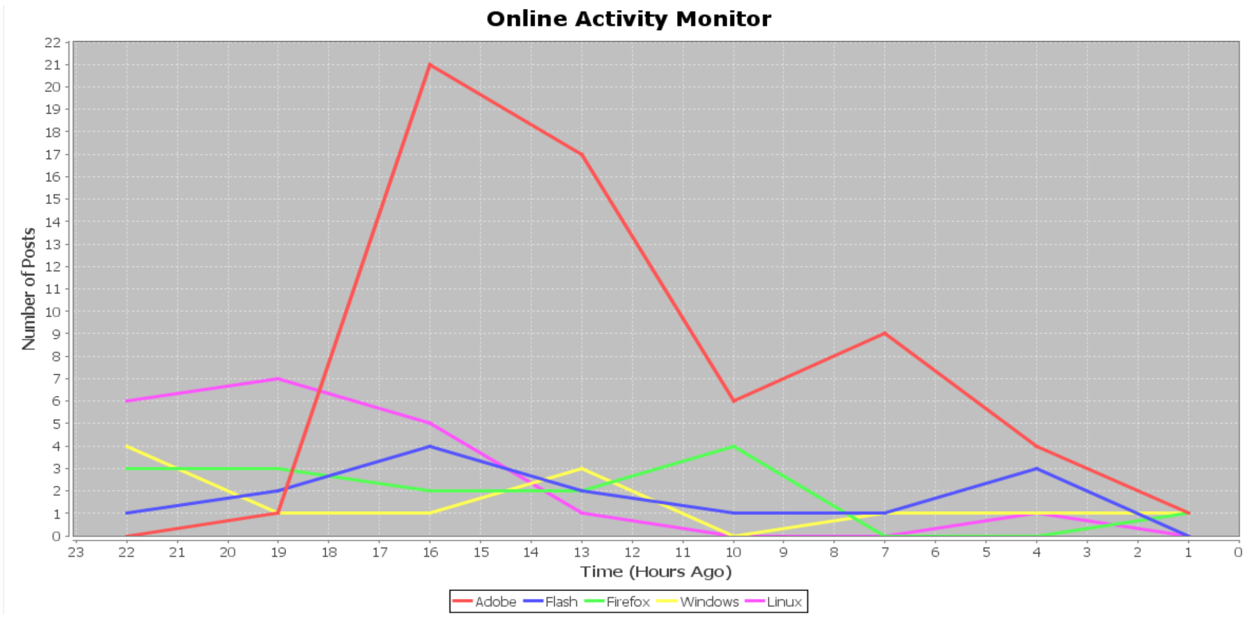


Figure 7: A spike in Adobe activity is easily noticed when displayed in a graph.

Another key feature of our prototype is the ability for analysts to see what others are doing, and be able to learn from the searches and refinements of previous searches. The prototype gave analysts the ability to see popular searches that other analysts are currently using. This gives analyst searches that are used the most representing current trends or searches that are constantly being used by analysts. The application also gives analysts a list of the most popular refinements to the search they are currently running. This ability allows analysts to learn from more experienced analysts and to refine their searches without having extensive knowledge of regular expressions. These functionalities successfully gave our prototype a social media aspect in which user are constantly learning from the actions of others and can work on protecting their network as a collective group. A small demo was also held to go over the functionalities of our User Interface with members of Group 51 including analysts and our thoughts that this social media aspect of learning and collaboratively working through our system was confirmed. It was also confirmed that a messaging type system would not be necessary in our system as all users must be working in the LRNOC here at Lincoln laboratory and will therefore be in the same room. Many other means of communication are already implemented inside the laboratory that would both be easier and more efficient to use than our system for any necessary communications. As whole our prototype was a successful proof of concept that using online, social media content to monitor cyber threats is possible. Going beyond the proof of concept a working prototype was delivered that allowed analysts to quickly and easily search gathered data and find information specific to their needs.

6 Conclusion

Over the nine working weeks at MIT Lincoln Laboratory the team leveraged the resources of Group 51 and ISD to accomplish the goal of creating an automated cyber threat aggregation and evaluation tool for analysts. Through the use of multiple interfaces, testing suites and standard documentation formats, the team accomplished the goals of a modularly designed, tested, and documented working prototype. While the project was scoped down from a fully automated threat detection system to a tool that give analysts the ability to detect these threats in real time, the initial goal of giving analysts the ability to more efficiently search over online content for relevant and important information was preserved. The project was also re-scoped to include a collaborative, team oriented aspect in which multiple analysts have the ability to work together and learn from each-others work. This change in focus did not change the direction of the project or prevent any of our initial goals from being accomplished, but rather added to the produced prototype in order to best serve potential users.

6.1 Future Work

While the prototype delivered is a fully functional system that analysts can use we feel that certain areas could still be improved given more time and resources. First, information retrieval was implemented in our prototype through regular expression matching. While this accomplished the goal of returning all posts pertaining to the given search, regular expression are very difficult and time consuming to write. Implementing more advanced information retrieval technologies could streamline this process for analysts and provide them with more information as to what topics are currently trending. More advanced systems could automatically detect spikes in certain topics and flag these topics as interesting, giving analysts the ability to look only at interesting topics.

As our system is used by more analysts we recommend that each analyst add sources to the Ingester that pertain to their own interests and needs. This will increase the amount and variety of information being pulled in and provide a more robust, powerful tool for analysts as it will always be pulling information they need. In addition, more source types (more Ingesters) will lead to a greater variety in posts (and the potential to add more sources and gather more information. For example, source types such as Facebook and even Ingesters specific to certain websites that are particularly accurate and up-to-date could prove to be especially useful. Each analyst that wants to use their system could add their own sources to the Ingester that specifically pertain to the information they are interested in, in order to make this tool more powerful for their own, personal needs. Adding additional sources will not threaten the core functionalities of our system or other analysts work as each search expression that is run only returns the posts that match that specific query.

To develop the system of recommending refinements upon particular regular expression searches, our system maintains a record of every refinement made upon a regular expression search. These refinements can be seen as edges in a larger graph of regular expressions. In our prototype, user see small sections of this graph as the refine their regular expressions based upon the most popular refinements made by other users of the system. With greater time and resources, the collection of all refinements can be used to develop a full weighted, directed graph showing all refinements that analysts made. This graph could be used to find the most useful regular expressions, as well as to study the process in which analysts make refinements to regular expressions. While this type of research is out of the scope of our project we feel that it is very interesting and due to the fact that the entire backend infrastructure is already set up, we feel that it is worth the time and resources.

This project leaves behind a fully working, tested and documented prototype for analysts to use as well as a full back end system that can be used in many other applications throughout the Laboratory. For example, all of our collected data is stored inside a MySQL database that can be easily accessed through our defined API. With minimal work other projects across the Laboratory can begin to use this data in applications of their own and use our API that has already gathered, processed, and stored the data for them. The LRNOC seems the most likely candidate for making use of this data as it is used as a tool to monitor the security of Lincoln's network. The information we gather also directly pertains to network security and could be used to protect the network before the vulnerability becomes a real threat to Lincoln Laboratory.

Appendix A: Database Design Tables

Transport Storage Database (SQLite)

Gathered Post Table

ID	Source Title	Source Type	Title	Author	URL	Date	Data
Primary Key, Auto-Increment							Unique

The Gathered Post Table is created by the Ingester and filled with raw posts pulled from the Web; after a pre-defined amount of time this table is shipped into the LRNOC where it is processed.

Source Table

ID	Source Type	Identifier	Source Title
Primary Key, Auto-Increment			

The Source Table is maintained by the Ingester and contains all of the sources that are currently being pulled from. Each source contains all of the information that the Ingester needs to successfully pull information from that source.

Source Type Table

Source Type
Primary Key

The Source Type Table is created by the Ingester from the source types (Ex: 'RSS' or 'Atom') defined in the Ingester. It contains all of the different types of sources that are supported. If additional source types are added they need to be added to the properties file so that the Ingester can add them to this table.

Permanent Storage(MySQL)

Post Table

ID	Source Title	Source Type	Title	Author	URL	Date	Data
Primary Key, Auto-Increment						Index	Unique(512)

Once the LRNOC side receives the Gathered Post Table from the Transport Database it adds all of the posts to the Post Table for permanent storage. The Date column is indexed to speed up queries against this table that are confined to a short time span. There is a unique constraint on the first 512 characters of the data field to ensure that no post is put into the database twice.

User Table

Username	Salt	Password
Primary Key		

The User Table is maintained by the LRNOC side. It keeps track of every user account and all necessary information to authenticate a user log in.

Dimensions Table

Dimension
Primary Key

The Dimension Table is maintained by the LRNOC side. It keeps track of ever supported search dimension. Dimensions are categorizations of loop-ups that can be done in the LRNOC including "Browser", "Operating System", "Domain", etc.

Regular Expression Table

ID	RegEx	Dimension
Primary Key, Auto-Increment	Unique	Foreign Key

The Regular Expression Table is maintained by the LRNOC side. It keeps track of every unique regular expression that is run. The Regular Expression string is constrained to be unique to ensure that each regular expression is only added to the table once. Each regular expression also has a dimension that defines what type of information this regular expression is searching for.

Saved Regular Expression Table

ID	Username	Name	RegEx
Primary Key, Auto-Increment	Foreign Key		Foreign Key

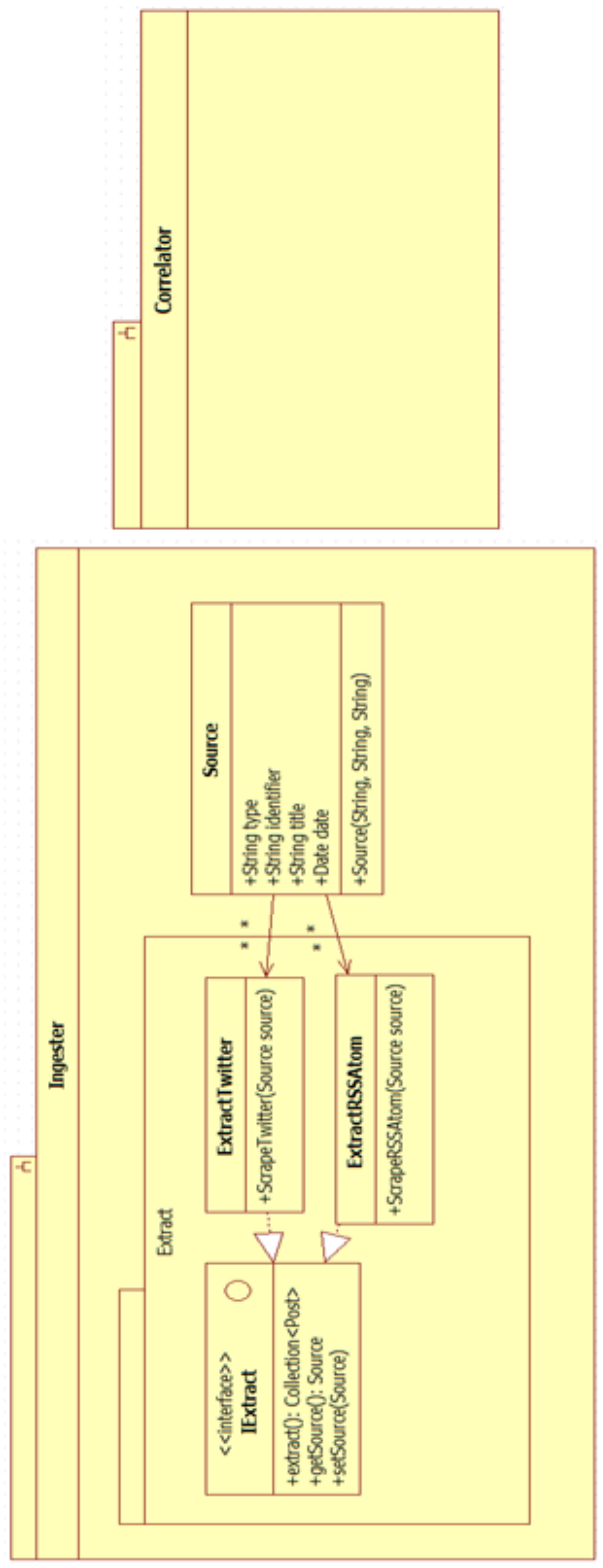
The Saved Regular Expression Table is maintained by the LRNOC side. It keeps track of every regular expression that a user saves. It also records who saved this regular expression and allows the user to give it a name so they can more easily find and reuse it. Multiple users can save the same regular expression.

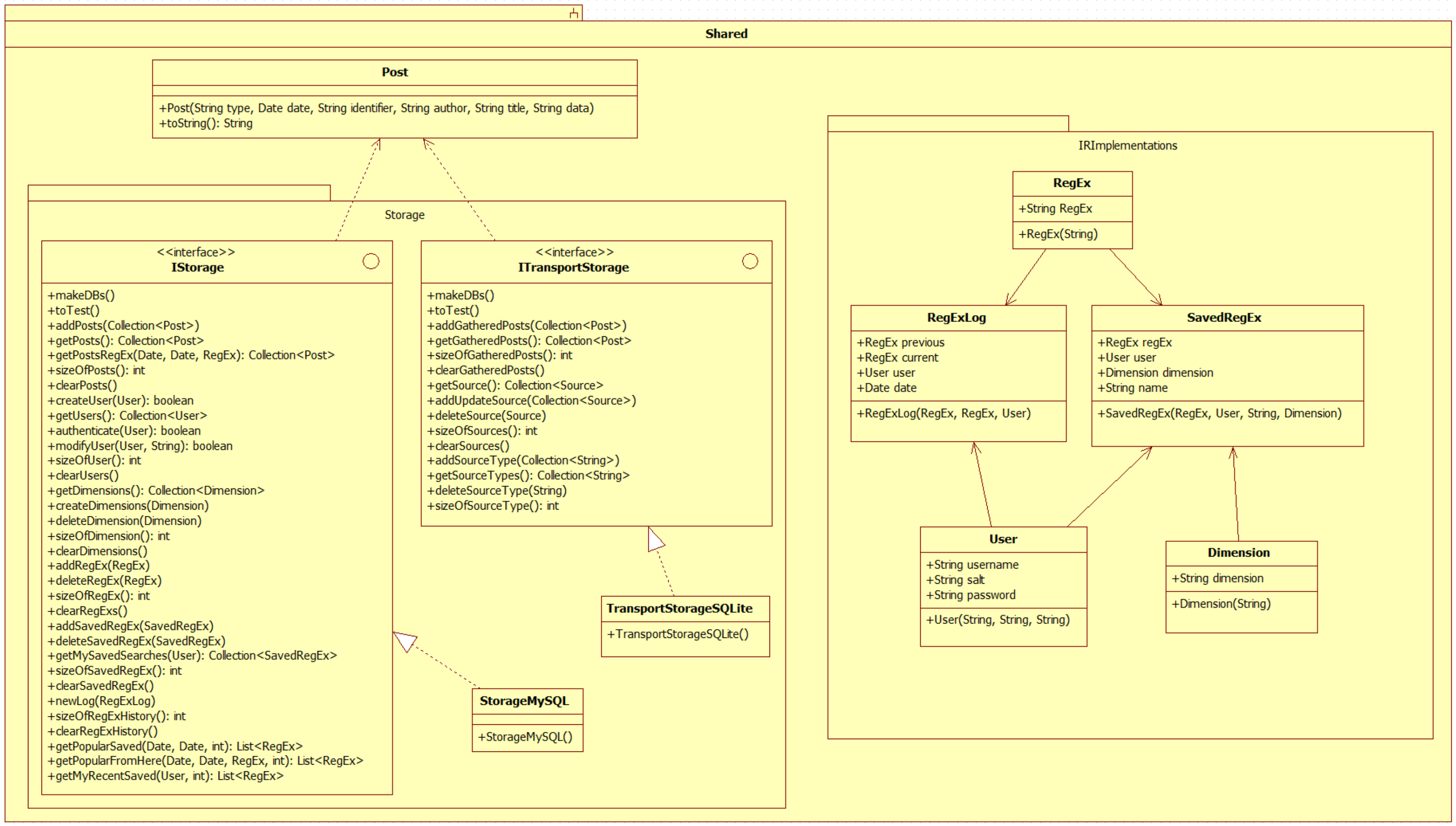
Regular Expression History Table

ID	Previous RegEx	Current RegEx	Username	Date
Primary Key, Auto Increment	Foreign Key	Foreign Key	Foreign Key	

The Regular Expression History Table is maintained by the LRNOC side. It logs every single regular expression search run. If this regular expression was the result of a refinement from another regular expression it also keeps track of what regular expression this one came from. A user and timestamp is also associated with each search logged in this table.

Appendix B: UML Diagrams




Shared
Post
+Post(String type, Date date, String identifier, String author, String title, String data)
+toString(): String
Storage
<<interface>>
IStorage
+makeDBs()
+toTest()
+addPosts(Collection<Post>)
+getPosts(): Collection<Post>
+getPostsRegEx(Date, Date, RegEx): Collection<Post>
+sizeOfPosts(): int
+clearPosts()
+createUser(User): boolean
+getUsers(): Collection<User>
+authenticate(User): boolean
+modifyUser(User, String): boolean
+sizeOfUser(): int
+clearUsers()
+getDimensions(): Collection<Dimension>
+createDimensions(Dimension)
+deleteDimension(Dimension)
+sizeOfDimension(): int
+clearDimensions()
+addRegEx(RegEx)
+deleteRegEx(RegEx)
+sizeOfRegEx(): int
+clearRegExs()
+addSavedRegEx(SavedRegEx)
+deleteSavedRegEx(SavedRegEx)
+getMySavedSearches(User): Collection<SavedRegEx>
+sizeOfSavedRegEx(): int
+clearSavedRegEx()
+newLog(RegExLog)
+sizeOfRegExHistory(): int
+clearRegExHistory()
+getPopularSaved(Date, Date, int): List<RegEx>
+getPopularFromHere(Date, Date, RegEx, int): List<RegEx>
+getMyRecentSaved(User, int): List<RegEx>
<<interface>>
ITransportStorage
+makeDBs()
+toTest()
+addGatheredPosts(Collection<Post>)
+getGatheredPosts(): Collection<Post>
+sizeOfGatheredPosts(): int
+clearGatheredPosts()
+getSource(): Collection<Source>
+addUpdateSource(Collection<Source>)
+deleteSource(Source)
+sizeOfSources(): int
+clearSources()
+addSourceType(Collection<String>)
+getSourceTypes(): Collection<String>
+deleteSourceType(String)
+sizeOfSourceType(): int
TransportStorageSQLite
+TransportStorageSQLite()
StorageMySQL
+StorageMySQL()
IRImplementations
RegEx
+String RegEx
+RegEx(String)
RegExLog
+RegEx previous
+RegEx current
+User user
+Date date
+RegExLog(RegEx, RegEx, User)
SavedRegEx
+RegEx regEx
+User user
+Dimension dimension
+String name
+SavedRegEx(RegEx, User, String, Dimension)
User
+String username
+String salt
+String password
+User(String, String, String)
Dimension
+String dimension
+Dimension(String)

Appendix C: Glossary

- **API**: a definition of how software components should communicate and work together
- **Compromised**: When a device is under (some amount of) control of a malicious party
- **Cyber Threat**: The possibility to gain unauthorized access to a system by exploiting a vulnerability in a host
- **Document** (Information Retrieval): One item of information in that a information retrieval technology searches over (Ex: a Twitter, or RSS post)
- **Exploit**: a piece of software created to take advantage of a vulnerability in a host
- **Host**: Any network enabled device
- **JavaDoc**: an API for formatting java documentation allowing for easy viewing and traversal
- **Modular**: a design strategy ensuring simple extension and modification of code without requiring design changes
- **One-Way Link**: a network protocol allowing for network traffic to flow in only one direction
- **Penetration Testing**: a method of evaluating host security by simulating attacks
- **Query** (Information Retrieval): a expression defining by a user describing the desired set of documents
- **Syndication Format**: a specified format in which information is represented
- **Vulnerability**: The inability of a device to withstand the attacks from a malicious sender

References

(2013, 9 23). Retrieved from Twitter4J Java Library: http://twitter4j.org/en/index.html

(2013, 9 20). Retrieved from ROME Java Library: https://rometools.jira.com/secure/Dashboard.jspa

(2013, 9 20). Retrieved from EclEmma Code Coverage: http://www.eclemma.org/

(2013, 9 15). Retrieved from JUnit Testing Framework: http://junit.org/

(2013, 9 7). Retrieved from MySQL Database: http://www.mysql.com/

(2013, 8 24). Retrieved from SQLite Database: http://www.sqlite.org/

Chang, C.-H., Kayed, M., Girgis, M., & Shaalan, K. (2006). A survey of web information extration systems. *Knowledge and Data Engineering, IEEE Transactions*, 1411-1428.

Greengrass, E. (2013). Information Retrieval: A Survey.

MITLL: Cyber Security and Information Science Accomplishments. (2013). Retrieved 2013 йил 23-09 from MIT Lincoln Laboratory: https://www.ll.mit.edu/mission/cybersec/cybersecaccomplishments.html

MITLL: Cyber Security and Informational Sciences. (2013). Retrieved 2013 йил 23-09 from MIT Lincoln Laboratory: http://www.ll.mit.edu/mission/cybersec/cybersec.html

MITLL: Cyber Systems and Operations. (2013). Retrieved 2013 йил 23-09 from MIT Lincoln Laboratory: http://www.ll.mit.edu/mission/cybersec/CSO/CSO.html

MITLL: Employment. (2013). Retrieved 2013 йил 22-09 from MIT Lincoln Laboratory: http://www.ll.mit.edu/employment/

MITLL: History. (2013). Retrieved 2013 йил 23-09 from Lincoln Laboratory: http://www.ll.mit.edu/about/History/history.html

MITLL: Organization. (2013). Retrieved 2013 йил 23-09 from Lincoln Laboratory: http://www.ll.mit.edu/about/Organization/organization.html

MITLL: Services Department. (2013). Retrieved 2013 йил 23-September from MIT Lincoln Labratory: http://www.ll.mit.edu/employment/services.html

Sunita, S. (2008). Information Extraction. *Foundations and trends in database 1.3*, 261-377.