University of Puerto Rico, Mayagüez Campus
Electrical and Computer Engineer Department

Network Anomaly Detection Suite
Final Report

Date:
May 16, 2016

Prepared for:
Nayda Santiago Santiago
Dr. Jose Vega Riveros
Dr. Isidoro Couvertier Reyes
Dr. Suresh Damodaran

Client:
Luis D. Lugo

Prepared by:
Marie A. Nazario (Project Manager)
Pedro A. Colon
Antoine A. Cotto

Executive Summary

Previously, the University of Puerto Rico at Mayaguez (UPRM) Electrical and Computer Engineering (ECE) Information Technologies (IT) department did not have the capability to monitor network anomalies in their network. Some of these anomalies, such as network loops and Secure Shell (SSH) and Simple Mail Transfer Protocol (SMTP) protocol attacks could have detrimental impact on the users of the network if they were left unchecked [10,17,20]. In order to fix these deficiencies the Grey Forest Security team developed the Network Anomaly Detection Suite (NADS). NADS is a configurable and extensible platform that is able to detect network loops and SSH/SMTP protocol attacks. NADS is able notify users through email and text messages of the detection of these anomalies and block offending Internet Protocol Addresses (IPs).

In order to meet our objectives the team has completed the research, design, implementation, integration, deployment, and documentation of the six modules that make up NADS. The first of the six modules is the Platform module. This module connects all the other modules in the framework together as well initializing all the necessary parts. The second module is the Algorithms module. This module holds both algorithm scripts that detect the network loops and the SSH and SMTP protocol attacks. The third module is the Data Retrieval module. This module is in charge of storing and abstracting the log data and returning it when queried. The fourth module is the Notifications module. The Notification module will take messages from the algorithm scripts and notify a user with these messages. The Visualization module contains and accesses the tool (Kibana [21]) that we will configure to show the data visualizations. The Utilities module handles the data ingestion formatting.

The team encountered no technical difficulties in the research and design phases of the project. The only setback we encountered was due to a misunderstanding of the necessary requirements of this document and in turn we had to push forward three planned tasks. In order to do this we had to work overtime to finish the modules and be ahead of schedule. The team began implementing the system on 3/4/2016 followed by the completion and integration of the modules on 4/27/2016. The system was deployed on 5/2/2016. The only current deadline is the final delivery of the project which is due on 5/16/2016.

The team expenditure as of this moment is of $69,878.46. As opposed to the projected expenditure of $71,986.67. As expected, the budget normalized even after the team has had to work overtime. The project concluded under-budget.

NADS will be provided as an open source tool. The software will be provided free of charge to users, however we will charge for the technical support (training, deployment and support) and application algorithm development of the product. Customers are encouraged to invest in the NADS, because other market tools come at expensive costs, are prohibitive on their extensibility, and do not offer the anomaly detections combination provided by the NADS.

Table of Contents

Introduction

The University of Puerto Rico at Mayagüez's Electrical and Computer Engineering (ECE) Department's network software only used to monitor network health (uptime, downtime, and node availability). The software however was not able to monitor network loops and protocol (SSH and SMTP) intrusion attempts. The lack of monitoring and notification of these network anomalies led to network downtime and email phishing attempts (attempts to steal personal information through emails) on the students and staff in the department [15]. To counteract this lack of detection, Grey Forest Security proposed the implementation of the Network Anomaly Detection Suite (NADS). NADS executes algorithms to detect network switch loops and, SSH and SMTP attacks. NADS then notifies users (i.e. system administrators) when one of these problems is detected so they could take action and reconfigure network switches or secure systems. NADS has been completed and is currently deployed in our client's system.

The NADS system requires some hardware and software prerequisites to run successfully. The software prerequisites in general include the installation of Java, Python, some prerequisite libraries and Python packages. These are prerequisites are named in more detail in the System Requirements appendix and the instructions on how to install them are provided in the User Manual. The system needs at least a 2GHz processor, and 2GB of ram to run the ELK stack and the loop detection and protocol detection algorithms. The exact Hardware requirements are provided in the System Specifications appendix.

From the initial proposal up until now, the team had three major variations. The first being the implementation of the Visualization module. Originally, the team had planned to develop, from the ground, up a complete visualization utility. However, after some research, the team found that it is possible to configure a dashboard in a utility called Kibana which would in turn do all of the visualizations. The second change concerns the Utilities Module. Once again, the team had planned on developing a complete formatting utility to ingest logs into the system. It was found that, since the team will utilize the ElasticSearch-Logstash-Kibana (ELK) stack [25], the team could do this through an Elasticsearch plugin. The last major change concerns the loop detection algorithm. For the loop algorithm, it was initially envisioned to be implemented using a trap based system. This plan was changed however when we found we could not set up custom traps on the switch, as such the algorithm was changed to a system that continually queries the switch's interface to see if they are being blocked by STP or not.

The sections that follow will describe NADS's design and specifications, a summary of the methods and approaches taken to reach the solution, a quick market overview, the results and impact of the project, a brief analysis of the expenditure undertaken in the project, and will end with our conclusions and future work that has to be done on the project. After this, comes a section of appendixes which contain detailed descriptions of the project design, process and impact.

Design Criteria and Specifications

The design alternatives for this project will now be discussed based on the project's six main modules. These modules are presented in the following figure:

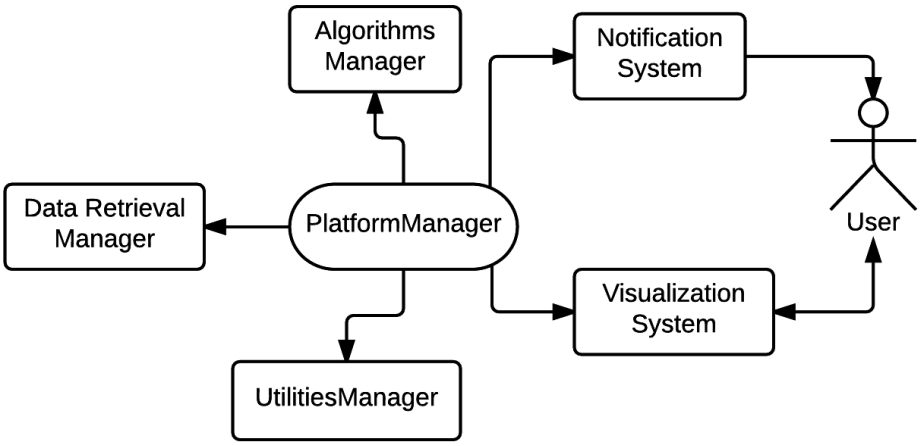


Figure 1: Overview of Project Modules

For the **Algorithmic module** the team had to deliberate over the alternatives for the SSH and SMTP attack detection algorithm and network loop detection algorithm. There were three alternatives for the SSH/SMTP attack detection algorithm. The first alternative was the Site-Wide Log-Based approach [16] which consists of modeling the typical site login failure distribution and monitoring changes to the mean of this distribution. The second alternative, the NetFlow/IPFIX based approach [28], utilizes specialized hardware to model a typical protocol attack assuming 3 attack phases and the Netflow/IPFIX traffic. The last approach consisted on averaging the daily login failures to the site and detecting when the current login failures surpassed this threshold. On the other hand, the network loop detection algorithm has two possible implementations. The first approach the team discovered consisted of the use of custom code mixed with the networking protocols STP and SNMP to lay software traps and detect and report a network loop when a trap is triggered. The second approach used physical and auxiliary data of the switch extracted from logs to establish a pattern of network events that when detected would indicate the possibility of network loop. A third approach was also researched later on in which instead of a trap based system we used an active query system. In this method we use python and SNMP call to continually query the switch to see the status STP status of each interface. Once an interface is detected to be blocked by STP we send a notification that a loop as has been found.

The **Data Retrieval module** had five main design alternatives that we will now discuss. The first alternative was to use the tool Splunk [7]. Splunk provided a way to ingest logs and parse the event data in them through the use of its proprietary algorithms. The second option was the use the ELK stack (a log parsing and ingestion system mixed with an efficient, API accessible, search engine) in conjunction with the Knowledge Query Language (KQL) [29] (an ontology based data abstraction layer that frees a developer from having to

worry about the physical storage of the data). The third approach involved directly accessing the logs and writing our own parsing and data mining algorithms that ran everytime we needed data. The fourth approach was using a database built by us that stored certain log data parsed by us. The final option involved using KQL to abstract the data we stored in a database as we previously described.

The **Notification** and **Utilities modules** alternatives will now be discussed. The Notification module had two possible alternatives. One option was the text message service Twilio [30] that allows text messages to be sent using a web API. The other option is using a Python SMTP library [31] to send emails and email based text messages. For the Utilities module the design alternatives were based on how to process the log files into a format acceptable by the Data Retrieval module. The first option was using a Logstash plugin called Grok [26] that came with various log parsing formats built in as well as the ability to create new ones. The second option was using custom code and regular expressions to build specialized formatting scripts.

Finally the **Visualization module** had four possible alternatives. The first approach, D3.js [23], is a JavaScript library for manipulating data that helps to bring data to life using HTML, SVG and CSS without tying yourself to a proprietary framework. The second option, vis.js [24], is an easy to use dynamic browser based visualization library. The third option, Bokeh [22], a Python interactive visualization library that targets modern web browsers, in the style of D3.js, and extends its capability with high-performance interactivity. Lastly the fourth option, Kibana, an open source analytics and visualization platform designed to work with ElasticSearch [32] and Logstash [33] that displays changes in logs through ElasticSearch queries in real time.

The analysis criteria we used to select our design alternative are described next. The first criterion that were established were based on our clients' constraints and needs. These constraints applied to the system and by consequence each module that composed it. The criterion established by the client were as follows. The system had to be free of charge as the ECE's IT department has no spare budget for software. The system had to be easily configurable. A user had to be able to configure options such as adding switches to be monitored or adding users to be notified. The network loop and SSH/SMTP attack detection had to be accurate and fast enough to detect the attacks as soon as possible. Finally the system should not use a large amounts of resources and it should not modify the original data found in the system. Finally we also added our own constraint in order to ensure the completion of all of our objectives. This constraint was the need for the data sources to be abstracted so that the framework could be made extensible to other types of algorithms and portable to use in other setups with very little configuration.

Based on these criterions, we decided which of the design alternatives were to be chosen for the individual modules. In the Algorithmic modules we chose the Site-Wide Log-Based approach for the SSH/SMTP attack detection algorithm. Initially for the loop detection we used traps based system however it was later changed to active query based system due to

some unforeseen circumstances with the hardware. We decided on the Site-Wide Log-Based detection for the attack detection algorithm because of it required no additional resources other than logs, and the ability to detect different types of attacks, as well as delivering all the event information (i.e. attack targets) the client wanted. The choice of algorithm for the network loop detection was chosen because of its high accuracy, speed of detection and configurability as well as compatibility with all the hardware. The data retrieval option we chose was the KQL- ELK stack combination. This combination provided a data retrieval framework that could abstract its data stores, the alternative was free of cost, and involved very little configuration of the retrieved data. For the Notification module the Python SMTP library was chosen. The SMTP library was the better choice for us as it was free of cost and supported both emails and text messages. Grok was chosen from the alternatives in the utilities block for the log pre-processing. Grok's plugin implementation with the chosen data retrieval implementation, as well as its built in support for popular log formats, gave us better ease of implementation and compatibility with the system.

The architecture of the system was designed to utilize our 6 main modules. These main modules are the Platform module, Algorithms module, Notification module, Visualization module, Data retrieval module, and the Utilities module. The Platform module will handle the initialization and lifecycle management of all the other modules, as such it must interface with them. The Algorithmic module will be in charge of managing the Python scripts that detect the anomalies. Both the switch loop and SSH/SMTP attack detection algorithms will interface with the Notification module to send the user email and/or text messages in the case that an anomaly is found. Once again, both detection algorithms will interface with the Data Retrieval module through log files generated by them to store the algorithms result so they may be viewed in the Visualization module. However only the SSH/SMTP attack detection algorithm will interface with the Data Retrieval module to actually retrieve data. The Data Retrieval module will also interface with the Utilities module by having it pre-process the logs for its ingestion. Finally the Visualization module will interact with the Data Retrieval module to get the necessary information as well as with the user so he or she can view and interpret the data. The following diagram shows the system architecture:

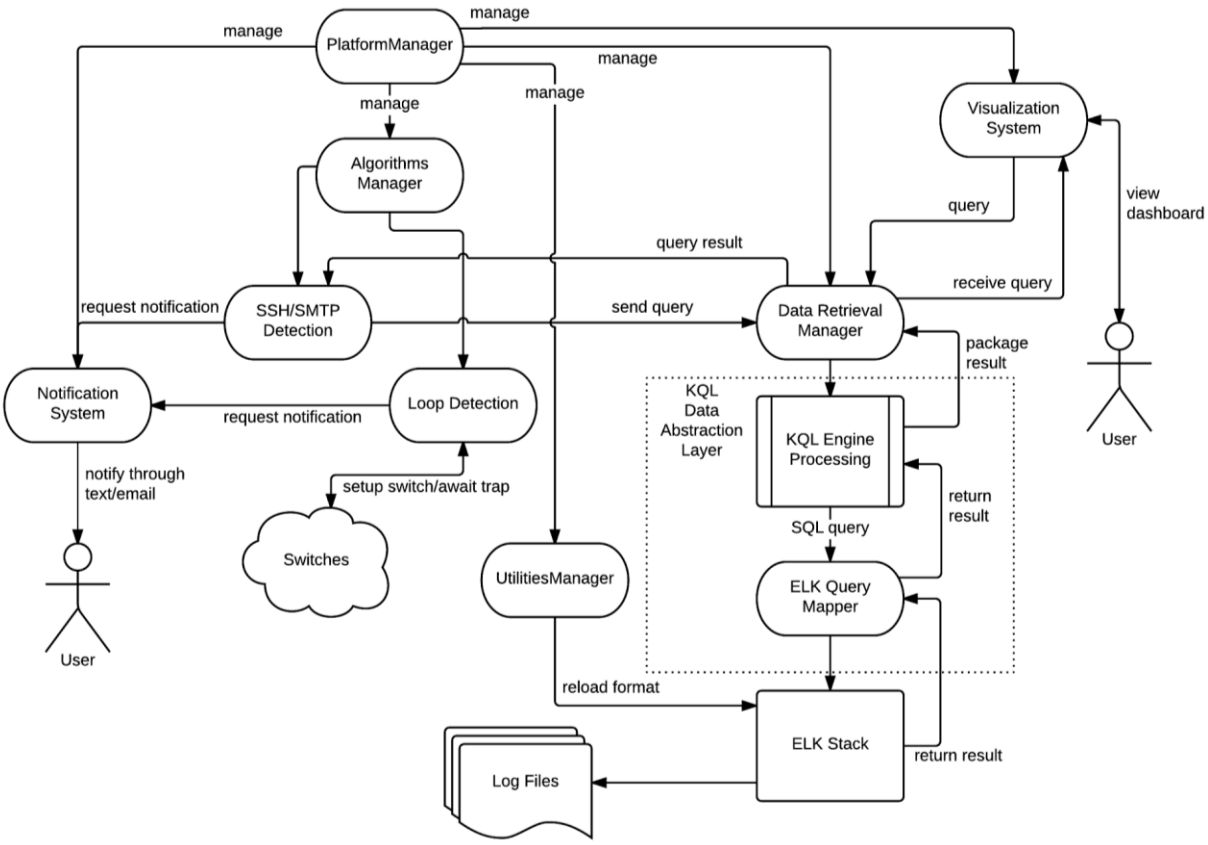


Figure 2: System Architecture Overview

After some schedule adjustments, the design, implementation and integration was completed for the six modules. Firstly, the design for the **Platform** module was carried out in an object oriented approach. The main platform manager class parses the configuration file that contains all of the options for the system. This class is in charge of passing the appropriate options to the individual module manager classes and starting them in their own thread of execution whenever possible. The individual module manager classes are in charge of interpreting their options and configuring themselves for correct execution. The module was implemented in Java using classes and methods. The module was then integrated piece by piece with the other modules. In each step of the process, after initial implementation and after integration with the modules, the module was tested using unit tests. The Data Retrieval manager class is in charge of receiving requests for data and serving them. This class abstracts the KQL - ELK layer and will also serve to respond to requests from the Visualization layer. The Notification Manager class is in charge of starting up the notification server which will listen to notification requests from algorithms and serve them accordingly. The Algorithm manager is in charge of passing the options necessary for the successful execution of the individual network anomaly detection algorithm and starting them in their own threads of execution. The Visualization Manager class is in charge of being able to load dynamic dashboards for the Visualization software. The Utilities manager is also in charge of being able to load dynamically different formatting configurations into Grok.

Secondly, the **Visualization** module design consisted of a prototype interface that was able to convey the information requested by the client (i.e. number of attacks, entities involved in the attack, login frequencies, and logins by period) and that could provide a searching mechanism for the user. The Visualization module was implemented in Kibana as a dashboard and some of its basic visualizations. Thirdly, the **Utilities** module consisted of designing an initial Grok regular expression to parse the data in log files to search for SSH/SMTP logins and ingest the data into fields. Grok was utilized to parse the protocol files and the result data.

Fourthly, the **Notification** module design consisted of a server algorithm that would continually listen for notification requests and interpret them to extract the request data and the request type (email or text or both). This design was then implemented in Python and tested through unit tests. The module was also tested after it was integrated with the algorithms and platform module. Fifthly, The **Data Retrieval** module design consisted of the development of a simple HTTP server in Java that would take data and write it to a file that is ingested by Logstash as a well as a way to query the ELK stack. To run queries on the ELK stack, a plugin called Elasticsearch-SQL was modified to parse KQL queries into SQL queries for execution. For this module after its implementation unit tests were created to make sure it properly wrote the data to the file that would be ingested as well as a simple query to make sure everything is returned properly. It was then integrated to the platform module so that the scripts would be able to access it.

Finally, the **Algorithm** module design consisted of the SSH/SMTP protocol attack and network loop detection algorithms. The SSH/SMTP attack detection was designed to be a Server-Client architecture [34] where a centralized server would interpret the data received from clients to determine if the mean of the distribution of failed logins had shifted and passed a threshold. The server would analyze the event that occurred and send out a request to notify users. Lastly, the network loop detection algorithm continually queries each switch to see the STP state of each interface in on the switches and see if anyone of them is blocked. When an interface is detected to be blocked the script notifies the user of the network loop with the pertinent information. Upon implementation both algorithms were first tested with unit tests and simulations of the anomalies. After integration both anomalies were once again simulated in order to verify the correctness of the algorithms.

Lastly, the team was able to complete an implementation of the ontology for usage with the KQL layer. The developed ontology is an essential part of the data abstraction system as it performs a mapping between the physical fields and their Dimensions and Tags. For a detailed explanation on the ontology and its design please refer to **Appendix K**.

In order for this system to be able to run correctly we need a processor of at least 2GHz, 2GB of RAM and at least 1GB of hard drive space. The system has been installed and tested on both Debian and Fedora systems with Python 2.7.X and Java 7 or greater.

For a detailed description of requirements, alternatives, diagrams, and designs view **Appendixes B, D, E and F**.

Methods and Approach to the Solution

The team organizational structure will be described now. Marie A. Nazario was the project manager, ensuring the successful completion of each stage and keeping track of the deadlines and work to do, on the other hand Antoine Cotto and Pedro Colon supported the project manager and helped with the system development. Marie was also in charge of the Visualization Module, the Notification Module, and contributed to the Loop Detection Module. Antoine was in charge of the Loop Detection and Data Retrieval Mechanisms. Finally, Pedro was in charge of the protocol attack mechanisms, and will contribute to the data retrieval mechanism, as he has worked with it in the past. All members were in charge of the Platform Module and Utilities module. Specifics of each team member work assignment can be found in the Gantt chart in Appendix I.

Figure 3: Team Organization and Work Distribution

For the creation of NADS we completed all tasks that were vital for the project. This included the Design of the platform system, algorithms (SSH and SMTP detection, and Loop detection), notification system, data retrieval system and visualization module. After all designs were completed we worked on the development and integration of each of the systems with the platform system, algorithms and the visualization tool. Finally we successfully deployed the system in our client system.

To complete these tasks the team ended up having to extend some of the tasks due to underestimating the work required. The team also had to move ahead some tasks due to misunderstandings with the required material for some of the milestones. This was counteracted by the team working more hours than anticipated to finish on time. On the other hand almost all milestones of the project were set back to compensate for changes in the initial milestones given by the course professors.

Throughout the process we developed automated JUnit tests for the individual modules in the platform system and PyUnit tests for the algorithms and notification system to ensure proper functionality. These tests were successful and showed that the modules worked with simulated cases. On the other hand, to test the proper integration of the system the team simulated attacks using Kali Linux and network loops with a physical switch in which two of the interfaces were connected, with the same cable, making a loop. These network anomalies were successfully detected in the tests, and notified the preconfigured users through text and/or email messages, and correctly logged the events into the Kibana dashboard.

Market Overview

The Network Anomaly Detection System (NADS) features a robust log ingestion system which was paired with a data abstraction mechanism that allows for the integration of additional algorithms as desired which results in an easily extendable platform. We leveraged this groundwork providing three specific detections: network loops at the physical layer, and network intrusion attempts through the SSH or SMTP protocols. As soon as the system detects either of these anomalies it will notify the configured user through a SMS and/or email message. This feature, the competitors, either do not have a default support or have to be extended to provide this.

Currently there is no one uniform tool that can offer a combination of the loop detection system, both of our intrusion detection capabilities, our notification system, and our data providing and expansion system. For data visualization, providing and analysis, the only current alternative is splunk [7] but with prices ranging from the $1800 per gigabyte a day or up to $600 per 100 gigabytes a day, it is very prohibitive for most small businesses. On the other hand, for the network intrusion detection, the only viable alternatives are tools like OSSEC [6] which offers solutions similar to ours (log analysis, time-based alerting, and active response), but does not include network health detection. AlienVault's Unified Security Management [11], on the other hand, has high prices as you scale the network up ($3900-$17,800) and proprietary detection schemes, and finally fail2ban [14] that only monitors ssh logins. Some of these platforms do not have much extensibility and their support for different log systems tends to be customized.

The NADS, with its open sourced components and extensibility, enters a market that has many useful but closed source solutions. With the introductory version we hope to provide an open-source tool that can be adapted for a business' specific algorithmic needs. The platform has a data abstraction layer that adapts to any sort of data that is attached to and still execute the same data processing algorithms successfully. Although the NADS software is free of cost, deployment and specific customer support such as the development of additional algorithms will be provided on demand for a price, in a manner similar to the commercial support model that Canonical Ltd. has. The commercial support model that the team adopted is based on selling annual support plans that adapts to the size of the enterprise we provide service to. In addition to this, we sell training on a per person or per group scale to enterprises. We also sell installation help for enterprises to facilitate their incorporation of the service.

The following is a table summarizing the alternatives to the problem stated.

	NADS	AlienVault USM	fail2ban	OSSEC	Splunk
Features	Network Loop Detection, SSH and SMTP (Distributed, and Brute Force) detection, Visualization	Asset Discovery, Vulnerability Assessment, Threat Detection, Behavioral Monitoring, Security information and event management, Visualizations	Abusive IP address detection and blocking of SSH and SMTP brute force attacks	Custom alert rules and writing scripts that take actions in response to security alerts.	Captures, indexes and correlates real-time data in a searchable repository from which it can generate graphs, reports, alerts, dashboards and visualizations
Extensible	Yes	No	Limited (In response to detection)	Limited	No
Price	Free	$3900-$17800	Free	Free	$1,800-$60,000 (per GB per month)
Source Code	Open	Closed	Open	Open	Closed

Table 1: Market alternatives to NADS

Results, and Impact of the Project

The team's technical results can be summarized by the successful implementation of all our system requirements. The team was able to implement a system that is able to detect both SMTP and SSH protocol intrusion attacks. The system was also able to detect network loops happening at the physical layer. The system was able to notify users upon detection of both anomalies through text and/or email. These modules were built upon a platform module that is made to be extensible to run other algorithms. This platform allows this extensibility through the use a configuration file which controls all the different settings in the modules such as users to be notified, algorithms to be run and their parameters, among other things. The system also uses an ontology based data abstraction system. The system currently abstracts a data store that uses the tools Logstash and Elasticsearch to ingest, store and query the logs necessary for the algorithms. The system also uses the tool Kibana to provide a dashboard to visualize all the results data of the algorithms.

The project's main ethical aspects can be summarized by the security of the users in the deployed network. This ethical conflict in our system is divided in two parts. One part is the possibility of blocking a user due to a false positive detection of an attack and the second part is the possibility that the system might fail. The solution we reached through the ethical analysis for the first conflict was a midway point between a very strict detection parameters and very lax detection parameters. This allowed us to properly protect the users while still maintaining a secure network. In the case that the system might fail we have no other choice but to try to make the system as foolproof as possible.

The legal aspects of the project are centered on the distribution rights. The project was made open source because of the BSD license that the data retrieval module uses. This BSD license also allows for the liberal distribution of the project as long as the license is provided beforehand. Thanks to this, the project will also have a BSD license for the distribution of the source code. This should free the team from legal problems due to distribution.

The social aspects of the project center on the safety of the student and employee population. The deployment of the project means that this population will be safer as attackers will be blocked and the downtime that network switch loops create will be minimized.

Budget Analysis

	Projected Expenditure	Total Expenditure
Human Resources	$ 41,496.90	$ 63,062.16
Equipment	$ 6,816.30	$ 6,816.30
Total:	$ 71,986.67 (w. overhead)	$ 69,878.46

Table 2: Labor Costs

The total expenditure on components, parts, software licenses, and other system resources amounted to $6,816.30. The team spent the entirety of the budget allotted for the Equipment. On the other hand, the total expenditure on personnel, consulting and other human resources amounted to $63,062.16. Most of the expenditure has come from the hours worked by the team, with the total amounting to $60,619.02. Due to some misunderstandings of the work required to accomplish deadlines resulting in having to put extra hours to deliver required material on time. The expenditure from the consultants comes out to a total of $2,443.14. The expenditures for the consultants was less than projected because the team did not need some of the consults as much as projected while others required more hours than previously projected.

Given the total expenditures sum for a total of $69,878.46, the team ended up with 1.45% over the projected budget without the overhead accounted, but with 1.03% under the projected budget with the overhead accounted. Taking this into consideration the team ended up using more than expected but still being covered by the overhead cost added to the allocated budget to the project.

For a detailed description of the project expenditures until the moment refer to **Appendix H**.

Conclusions and Future Work

NADS was designed and implemented successfully. The individual modules of the system were implemented based on the design that was proposed on the progress report. There were some slight deviations from the original design such as the usage of the elasticsearch-sql in the Data Retrieval module, but these did not change the pre-designed functionality of the modules. After the modules were completed, integration of the system began as planned. The integration was completed successfully and the system was tested to perform the loop detection and protocol attack detection and their respective notification and visualization.

The initial implementation of the NADS project presents an extensible platform that can be utilized to run network security algorithms. This initial release is intended to serve as the groundwork for a more secure internal network for the ECE department. We hope that this release will help the student body to have more security by preventing possible compromisations of the SMTP accounts (and in turn spam emailing) and system accounts (and in turn data theft/loss). NADS will also serve as a tool that will allow, in the future, for more secure networks as it will allow for future implementation of other more complex monitoring/prevention algorithms. The open-source and free nature of the system means that it can be utilized in places with limited budget for security mechanisms. It also means that the system can be extended by other people to improve or fix aspects of the system.

The design and implementation of the project served to teach the team how to work with the SNMP protocol and how to work with deriving algorithms from scientific papers. The project also taught the team that deployment is much harder than imagined and more time should be allocated to it in future projects. The team also learned that hard work pays off as once the system was deployed, it was able to prevent protocol attacks from unknown IPs originating from places such as Russia.

This is an initial release of the system. Future modifications to the system should focus on making the system more user friendly and developing more intuitive graphical user interfaces for the presentation of results and configuration of the system. Other modifications that can be done are the addition of other NAD algorithms to improve the security of the monitored network/machines.

Bibliographic References

1. Adkins, W. (n.d.). How do I Calculate Overhead Costs? Retrieved January 25, 2016, from http://smallbusiness.chron.com/calculate-overhead-costs-3481.html
2. EC2 Instance Pricing – Amazon Web Services (AWS). (n.d.). Retrieved January 25, 2016, from https://aws.amazon.com/ec2/pricing/
3. Hansteen, P. (n.d.). The Hail Mary Cloud and Lessons Learned. Retrieved May 10, 2013, from http://bsdly.blogspot.de/2013/10/the-hail-mary-cloud-and-lessons-learned.html
4. Integration Testing. (n.d.). Retrieved January 25, 2016, from https://msdn.microsoft.com/en-us/library/aa292128(v=vs.71).aspx
5. Learn About Scrum. (n.d.). Retrieved January 25, 2016, from https://www.scrumalliance.org/why-scrum
6. OSSEC. (n.d.). Retrieved January 25, 2016, from http://ossec.github.io/
7. Operational Intelligence, Log Management, Application Management, Enterprise Security and Compliance | Splunk. (n.d.). Retrieved January 25, 2016, from http://www.splunk.com/
8. Overhead Definition | Investopedia. (2003, January 24). Retrieved January 25, 2016, from http://www.investopedia.com/terms/o/overhead.asp
9. Professors in the United States. (n.d.). Retrieved January 25, 2016, from https://en.wikipedia.org/wiki/Professors_in_the_United_States
10. Shackleford, D. (2008, December 1). Real-Time Adaptive Security.
11. Unified Security Management (USM) Platform. (n.d.). Retrieved January 25, 2016, from https://www.alienvault.com/products
12. Unit Testing. (n.d.). Retrieved January 25, 2016, from https://msdn.microsoft.com/en-us/library/aa292197(v=vs.71).aspx
13. Work Breakdown Structure (WBS). (n.d.). Retrieved January 25, 2016, from http://www.workbreakdownstructure.com/
14. Fail2ban. (n.d.). Retrieved January 25, 2016, from http://www.fail2ban.org/wiki/index.php/Main_Page
15. L. Lugo, Personal Communication, January 2016.
16. Javed, M., & Paxson, V. (2013). Detecting stealthy, distributed SSH brute-forcing. Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security - CCS '13.
17. Nickerson, J. (1993, May 24). Client/Server chaos yields valuable lessons. Network World, 41-44.
18. Owens, J., & Matthews, J. (2008). A Study of Passwords and Methods Used in Brute-Force SSH Attacks.
19. Fringe Benefits Definition | Investopedia. (n.d.). Retrieved February 12, 2016, from http://www.investopedia.com/terms/f/fringe-benefits.asp#ixzz3zpCcFGse

20. S.U, U., O, O. C., & C.O, O. (2015). Tracking the Effects of Loops in A Switched Network Using Rapid Spanning Tree Network. *International Journal of Research in Electronics and Communication Technology, 2*(3), july-sept. 2015, 10-15.
21. Kibana. (n.d.). Retrieved February 25, 2016, from https://www.elastic.co/guide/en/kibana/current/introduction.html
22. Welcome to Bokeh. (n.d.). Retrieved February 25, 2016, from http://bokeh.pydata.org/en/latest/
23. D3.js - Data-Driven Documents. (n.d.). Retrieved February 25, 2016, from https://d3js.org/
24. Vis.js. (n.d.). Retrieved March 01, 2016, from http://visjs.org/
25. An Introduction to the ELK stack | Elastic. (n.d.). Retrieved March 01, 2016, from https://www.elastic.co/webinars/introduction-elk-stack
26. Grok. (n.d.). Retrieved March 01, 2016, from https://www.elastic.co/guide/en/logstash/current/plugins-filters-grok.html
27. Simple Network Management Protocol. (n.d.). Retrieved March 01, 2016, from https://en.wikipedia.org/wiki/Simple_Network_Management_Protocol
28. Hofstede, R., Hendriks, L., Sperotto, A., & Pras, A. (2014). SSH Compromise Detection using NetFlow/IPFIX. *ACM SIGCOMM Computer Communication Review SIGCOMM Comput. Commun. Rev., 44*(5), 20-26.
29. Damodaran, S. K. (2016). *Knowledge Query Language* (Rep. No. 1999). Lexington, MA: MIT Lincoln Laboratory
30. Twilio. (n.d.). Retrieved March 01, 2016, from https://www.twilio.com/
31. Smtplib — SMTP protocol client. (n.d.). Retrieved March 01, 2016, from https://docs.python.org/3.4/library/smtplib.html
32. Elasticsearch 2.0 Overview & Demo | Elastic. (n.d.). Retrieved March 01, 2016, from https://www.elastic.co/webinars/elasticsearch-2-0-overview?elektra=home
33. Logstash | Elastic. (n.d.). Retrieved March 01, 2016, from https://www.elastic.co/products/logstash
34. Client Server Model. (n.d.). Retrieved March 01, 2016, from https://en.wikipedia.org/wiki/Client–server_model
35. What is distributed denial-of-service attack (DDoS)? (n.d.). Retrieved March 01, 2016, from http://searchsecurity.techtarget.com/definition/distributed-denial-of-service-attack
36. What is brute force cracking? (n.d.). Retrieved March 01, 2016, from http://searchsecurity.techtarget.com/definition/brute-force-cracking
37. Switching Loop. (n.d.). Retrieved March 01, 2016, from https://en.wikipedia.org/wiki/Switching_loop

38. Daemon (Computing). (n.d.). Retrieved March 01, 2016, from https://en.wikipedia.org/wiki/Daemon_(computing)

39. Ontology. (n.d.). Retrieved March 01, 2016, from https://en.wikipedia.org/wiki/Ontology_(information_science)

40. What Is a Socket? (n.d.). Retrieved March 01, 2016, from https://docs.oracle.com/javase/tutorial/networking/sockets/definition.html

41. Python. (n.d.). Retrieved March 01, 2016, from https://www.python.org/

42. Introducing JSON. (n.d.). Retrieved March 01, 2016, from http://www.json.org/

43. The Official YAML Web Site. (n.d.). Retrieved March 01, 2016, from http://yaml.org/

Appendix A: Glossary

This section contains the definition of some acronyms and terminology used throughout the document.

Acronyms:

ELK - ElasticSearch, Logstash, and Kibana
NAD - Network Anomaly Detection
NADS - Network Anomaly Detection Suite
SSH - Secure Shell
SMTP - Simple Mail Transfer Protocol
SNMP - Simple Network Management Protocol
OS - Operating system
KQL - Knowledge Query Language
JVM - Java Virtual Machine
API - Application Programming Interface
SQL - Structured Query Language
HTTP - Hyper Text Transfer Protocol
REST - REpresentational State Transfer

Terminology:

Distributed Attack - A multitude of compromised systems **attacking** a single target [35]

Brute-force Attack - Brute force (also known as brute force cracking) is a trial and error method used by application programs to decode encrypted data such as passwords or Data Encryption Standard (DES) keys, through exhaustive effort (using brute force) rather than employing intellectual strategies. [36]

Switch Loop - A Switching loop or bridge loop occurs in computer networks when there is more than one Layer 2 (OSI model) path between two endpoints (e.g. multiple connections between two network switches or two ports on the same switch connected to each other). [37]

Service/Daemon - In multitasking computer operating systems, a daemon[] is a computer program that runs as a background process, rather than being under the direct control of an interactive user. [38]

Ontology - In computer science and information science, an ontology is a formal naming and definition of the types, properties, and interrelationships of the entities that really or fundamentally exist for a particular domain of discourse. [39]

KQL	- Knowledge Query Language: Data abstraction mechanism that uses the concepts of Dimensions, Tags, and Dimension Sets to abstract physical data fields
Dimension	- An immutable aspect of data. Essentially the type of the data.
Tag	- A mutable aspect of data. Essentially the meaning or context that data has.
DimensionSet	- Set of Dimensions.
Sockets	- A *socket* is one endpoint of a two-way communication link between two programs running on the network. A socket is bound to a port number so that the TCP layer can identify the application that data is destined to be sent to. [40]

Appendix B: User Requirements

This section contains the functional and nonfunctional user requirements. These requirements were generated based on conversations with the client and research done to address the client's needs. The requirements were verified with the client to avoid misunderstandings.

A. Functional

- Essential

1. The system will be able to detect SSH intrusion attempts.
2. The system will be able to detect SMTP intrusion attempts.
3. The system will be able to detect network loops at the physical layer.
4. The system will be able to send notification emails to pre-configured emails of SSH/SMTP intrusion attempts.
5. The system will be able to send text notifications to pre-configured phone numbers of SSH/SMTP intrusion attempts.
6. The system will be able to send email notifications to pre-configured emails of network loops.
7. The system will be able to send text notifications to pre-configured phone numbers of network loops.
8. The system will ban IP's detected to be attacking the node.
9. The system will run in the background (preferably as a service).
10. The system will be able to process ingested log data through an ontological abstraction created by a user.
11. The system will be able to provide situational information on what caused a notification to be sent.
12. The system will be able to visualize relevant log data and events to the user
13. The system will be able to ingest many log types with a simple parsing/preprocessing script.
14. The user will be able to set up parameters to the different algorithms through the use of a configuration file.
15. The user will be able to edit the following information in a configuration file:
    a. Phone numbers and emails of persons to be notified.
    b. Folder location of algorithms to be run.
    c. Parameters for each algorithm to be run.
    d. Running frequency of the algorithms.

- Desirable
  1. The system will have a simple command line interface to allow easier control of the operations it performs.
  2. The system will perform in-depth analysis of SSH and SMTP attacks to provide information on whether it was a distributed or singleton attack, and provide information on the attack target.
  3. The system will contain a visual interface for building KQL queries to permit for data-store independent searches.
- Optional
  1. The system will have a visual configuration utility.

2. Non-functional

- Usability
  1. The system will be able to be set up with as few as 3 commands after the initial setup.
  2. The system will be designed for the use of system administrators.
- Reliability
  1. The system will be able to execute only the algorithms that are configured correctly and nothing more.
  2. The system will be able to terminate the algorithms it is executing in the case that the system needs to be shut down.
  3. The system will provide helpful, detailed error messages.
- Security
  1. The system will not modify an operating system in a destructive manner.
  2. The system will not contain any type of login data that may be used to gain unauthorized access to network hardware.
  3. The system will verify the integrity of the configurations of algorithms that it will execute in order to not execute malicious routines.
  4. The system shall encrypt messages passed through its modules.
- Interface
  1. The system will be able to effectively interact with the authentication logs of a Debian Linux installation.

2. The system will be able to interact with other installations of the system in other nodes to be able to acquire information that may be necessary for the successful execution of an algorithm.
3. The system will be able to interact with networking protocols such as SNMP to gain necessary information.

- Performance
  1. The system will be able to identify network loops, SSH attacks and/or SMTP attacks, within five minutes of the event.
  2. The system will be able to notify configured users within five minutes of the detection of an event.
  3. The system platform will not use an excessive amount of resources so as to render a host machine useless.

Appendix C: System Specifications

The initial minimum hardware requirements for the system are as follows:

System Requirements	
Processor	2GHz or faster
Memory	2 GB of DDR3 RAM or more
Hard Drive	1 GB of disk space or more
Operating System	Ubuntu Linux
Software	Java 7 or greater, Python 2.7, Filebeat 1.1.1, PHP 5, ElasticSearch 2.2.0, Kibana 4.4.1, Logstash 2.2.2-1
Networking Hardware	STP, SNMP, and SSH enabled switch
Other needs	Internet Access is required for text and email notification.

Table 3: Minimum System Requirements

Based on the user requirements, the system is able to:

- Notify users when the following anomalies are detected in the monitored network and systems:
  - Network Switch Loops
  - SSH and SMTP Attacks
- The notification that the system send to a user is in the form of a text and/or an email message through a Python SMTP Library.
- The notifications for the SSH and SMTP attacks contain information such as the attack target, the attack source, and the type of the attack.
- The notifications for the loop detection contain information such as the problematic switch
- The system has a platform framework that permits the execution of pre-configured Network Anomaly Detection (NAD) algorithms written in Python [41].
- The system utilize a data abstraction mechanism combined with a formatting utility to permit easy algorithm development that relies on system data. The data abstraction mechanism was done using the KQL system combined with the ELK stack for log management, parsing, and indexing.
- The system runs in a daemon thread in the background to perform its detection activities continuously
- A user is able to edit a JSON [42] configuration file to configure the algorithms they want to execute, the users they wish to notify, and other parameters such as directory locations.
- The system utilize Kibana to perform query and visualization activities by a user.

Appendix D: Analysis of Alternatives

The following section discuss the alternatives presented for the system and the reason behind the alternatives chosen.

System Alternatives:

The client needed a system that could be extensible, independent of the stored data format, and that could run SSH/SMTP attack detection, through logs, and network loop detection. The team needed a system that would be of little to no cost as the client has no budget for software licenses. The team also wanted a tool that would not be tailored to a specific use case and could be tasked with multiple varied routines.

Criteria	**NADS**	**OSSEC** [6]	**fail2ban** [14]	**Splunk**	**AlienVault USM** [11]
Extensibility	Independent, configurable algorithm script execution	Customizable Alerting and scripting based on response to alerts	Arbitrary actions can be configured in the case of a malicious event detection	Dashboard and search oriented API.	Customized Data Input for the system
Source Code	Open	Open	Open	Closed	Closed
Price	Free (Paid Support)	Free	Free	Scalable Data Usage Pricing	Scalable Machine Count Pricing
Detections	SSH/SMTP Attack, Network Loop	File integrity log monitoring, root check, and process monitoring	Malicious IP Logins	Customizable according to available data	Network IDS, Host IDS, File Integrity Monitoring (FIM)
Purpose	General network anomaly detection	System Wide Monitoring	Malicious Log and rule Based Activity detection	Data Capturing, analysis and visualization	Unified threat detection and compliance management solution

Table 4: Comparison of System-Scale Alternatives

After consideration of the current systems, the team chose to develop NADS. Through the research conducted into our problem's possible solutions and our market analysis, we found that currently there was no one alternative that can offer the full range of features that NADS offers. Due to our client's need of a free tool the Splunk and AlienVault system were eliminated from consideration due to their price tag. OSSEC and fail2ban were discarded from the process because of their limited range of detections

and extensibility. Since the available system alternatives did not meet the client's needs the team opted to make a platform tailored to our customers' specifications (NADS).

Platform and Configuration Alternatives:

The team build the system from the ground up. The team required a framework that could execute all of the individual modules independently from each other and could initiate a simultaneous shutdown action. This platform also ingest a configuration file and pass the corresponding options the modules that they belong to.

Criteria	JSON	YAML [43]
Language Support	High (Most languages contain an implementation of a JSON parser)	Low (Some languages contain YAML parser, but are not up to date)
Comment Support	No	Yes

Table 5: Comparison of Configuration format alternatives

Since the platform was so specialized in its purpose, the team developed it from the ground up. This way they had fine control over the parsing and initialization processes. However, with regards to the format of the configuration file that the team used, the team decided to use a JSON formatted configuration file. This format is human readable, and is supported by very stable and up-to-date parsers in both languages (Python and Java) that the team used. The only downside from selecting this format and not YAML, was losing the ability to provide inline comments. This was remedied by a text file provide that explains the possible configuration options.

Visualization Alternatives:

The client desired a way to visualize the results of the algorithms that the platform executes, and parts of the data that the algorithms utilize. The client wanted the visualization utility to be able to work with the least configuration possible. The team also evaluated aspects such as ease of implementation and extensibility for future visualization requirements.

Criteria	Kibana	D3	Bokeh	vis.js
Ease of Implementation	Existing software client. Requires initial setup only. Kibana was developed as part of ELK stack	Easy to incorporate, yet requires the development of a web application.	Requires the implementation of a Python application	Easy to implement but also requires the development of a web application
Ease of tool configuration	Low-Medium	High	Medium	High

Extensibility	Low-Medium	High	High	High

Table 6: Comparison of Visualization Alternatives

Taking into consideration the criterion, the team utilized Kibana. Kibana was simple to integrate with the data retrieval mechanism that was used (Logstash and ElasticSearch). Additionally, since Kibana was an already developed product the team only need to add a few extensions and dashboards to configure it to the client's needs. The other three alternatives required building a visualization app from scratch and formatting data to tailor it for the alternatives' ingestion mechanisms. Lastly, Kibana requires little user expertise to be able to operate; the user just needs to open the Kibana web interface through a web browser.

Utilities Alternatives:

The team needed a way to take the log data generated by the network and process it into a format accepted by Logstash. We needed this to be done in a way that would be simple enough that users could do it without extensive text parsing and processing experience.

Criteria	Logstash plugin (grok)	Building of code and regular expressions
Support for various log formats	High (Comes with built in support for most type of log formats and fields)	High (Since everything will be custom built for the needed logs, it is easier to support diverse kinds of logs)
Extensibility	High (Adding an additional pattern is as simple as writing it and reloading it into Logstash)	Medium (Adding additional log types requires the development of new regular expressions)
Compatibility	Dependent on the availability of Logstash in the system	Dependent on availability of Python in the system
Ease of implementation	High (Simple pattern file)	Low (Need to write a Python script that analyzes and formats the log data)

Table 7: Comparison of Utilities Alternatives

After concluding the research, the team decided to use the Logstash plugin called Grok. Grok was chosen over a custom Python parsing script as Grok proved easier to implement. Grok already had built-in support for various different log patterns as well as being made for Logstash, it made it easier for future users to add support for new log files.

Algorithmic Alternatives:

- SSH/SMTP Detection

  The client requested for the team to implement a way to detect malicious login attempts to the SSH and SMTP protocols. The client also specified that the algorithm should detect the attack as soon as possible and notify the system administrator of the presence of the attack, the target, and the source of the attack.

Criteria	Site-Wide Log-Based approach (Host Based Approach)	NetFlow/IPFIX Approach (Flow Based approach)	Login Threshold (Host-Based Approach)
Ease of Implementation	Intermediate-Complex (Contains relatively straightforward implementation aspects, yet requires analysis to tailor to a specific network)	Complex (Requires specific hardware setup and a software system to be able to analyze data and fit it to a statistical model)	Simple (Analyze login patterns to establish daily thresholds)
Dependencies	Log Data from Nodes	Hardware to analyze network flow data	Log Data
Accuracy	Medium-High (Possibility of classifying a legitimate user as an attacker)	High (Very low possibility of misclassifying the event of an attack)	Low (High possibility of misinterpreting a busy day with an attack)
Classification	Singleton, Distributed Attack	Presence of an attack	Presence of an attack
Price	Free	Cost of hardware	Free
Event Information	Source, Target, Protocol, Attack Type	Attack Presence	Attack Presence

Table 8: Comparison of Alternative SSH/SMTP Algorithms

After a literature review, the team opted for the Site-Wide Log and Host-Based approach. This approach was chosen after eliminating the flow based approach due to our budget restriction and the Login threshold approach due to not having the event information our client required and its lack of accuracy. The Site-Wide Log-Based approach also gave us the ability to detect individual and distributed protocol attacks, a feature that distinguishes it from the other alternatives.

- Loop Detection

  For the loop detection algorithm the client required that the loops be detected as soon as possible and that we could give him event information on the affected switch. He also required a way to control the switches that were covered by the algorithm. Additionally we aimed to have a system that could accurately detect the loops as well as being as compatible with different switches as possible.

Criteria	STP/SNMP/custom code	Physical and auxiliary data detection
Ease of implementation	Medium (Implementation is helped through the use of existing tools and protocols)	Complex (Involves the development of thresholds, models, and rules that indicate a loop may be happening)
Compatibility	Medium (Compatibility is lessened by the different command syntax of switch models)	Medium (Compatibility is lessened by syntax difference in logs of different switch models)
Accuracy	High (Uses IEEE protocol to detect loop)	Low-Medium (Uses log data to try to detect if there's a loop)
Configurability	High (Allows us to select the switches by adding them to a configuration file)	Low (No way to remove or add switches besides parsing and removing log data)
Event information	Generates switch system information	Generates switch system information
Speed of detection	Fast (As soon as a switch is connected to another switch we can tell if a loop happened)	Slow (We have to wait for various symptoms of the loop to occur repeatedly before being able to determine if a loop is happening)
Dependencies	Requires STP, SNMP and SSH support by switch	Requires log and device state data

Table 9: Comparison of Alternative Loop Detection Algorithms

After going through the options, the team decided to use a combination of custom code and access to network protocols (STP/SNMP). The team chose this option because it met all of our client's and all of the team's requirements. It gave the best accuracy of detection as well as being faster at detecting the problem than the other alternatives. Not only that, it gave the team better configurability as well as being simpler to implement. Both alternatives were even in compatibility and in the amount event information generated.

Data Retrieval Alternatives

The client required that the software was capable of running with as less configuration as possible with the existing data in the department servers. In addition to this, the team desired that the algorithms that the platform would execute be independent of the storage and format of the available data. The team also desired that the mechanism would retrieve the data as precisely, with as little modification as possible. The client also desired that the system have the lowest cost possible.

Criteria	Splunk	KQL - ELK	Direct Log Access	KQL - Database	Direct Database
Data Store Independent	No	Yes	No	Yes	No
Ease of data access	High	Medium	Low-Medium	Medium	High
Ease of implementation	Medium-High	Medium	High	Medium	Low
Price	Commercial license	Free	Free	Free and available with Paid Extensions	Free, commercial options available

Table 10: Comparison of Alternative Data Retrieval Methods

After some research, the team opted to use a combination of KQL with the ELK stack. This combination met our needs of being free of cost, and being independent of the physical storage (tables, categories, etc.) of the data. It also met the criteria of being able to work with little modification of the existing data. The selected combination also had major elements already implemented which decreased the complexity of the implementation. This was in contrast to the other alternatives that did not meet these criteria. The direct log access and direct database solutions were discarded as they were not data store independent and required extensive manipulation of the system data. The splunk alternative was discarded due to not meeting the client's price requirement. The KQL and database alternative was discarded as this would essentially result in duplicated data that had to be carefully monitored for collisions and maintained whenever new data was added.

Notification Alternatives

The client needed a system that could notify, in a real-time fashion, an intrusion attempt or loops detected by the algorithms in the system through text messages and emails. The system needed to be of no cost and have as less dependencies as possible. Finally, the team also desired for the mechanism to be easy to implement.

Criteria	Twilio	Python SMTP Library
Email messages	No	Yes
Text messages	Yes	Yes
Ease of implementation	Simple	Simple
Dependency	Need to create an account in the service.	None

Cost	Fee per text message	Free
Security	Could not guarantee security	Yes

Table 11: Comparison of Notification Alternatives

Considering the criteria mentioned the team chose the Python SMTP library. Twilio was too expensive for the client and just provided support for text messages. Finally, with Twilio, we could not guarantee the security of the user's information as the message is sent through a third party system. Both alternatives were tied in the simplicity of their implementation as a small tool had to be developed for both to work.

Appendix E: System Architecture and Interfaces

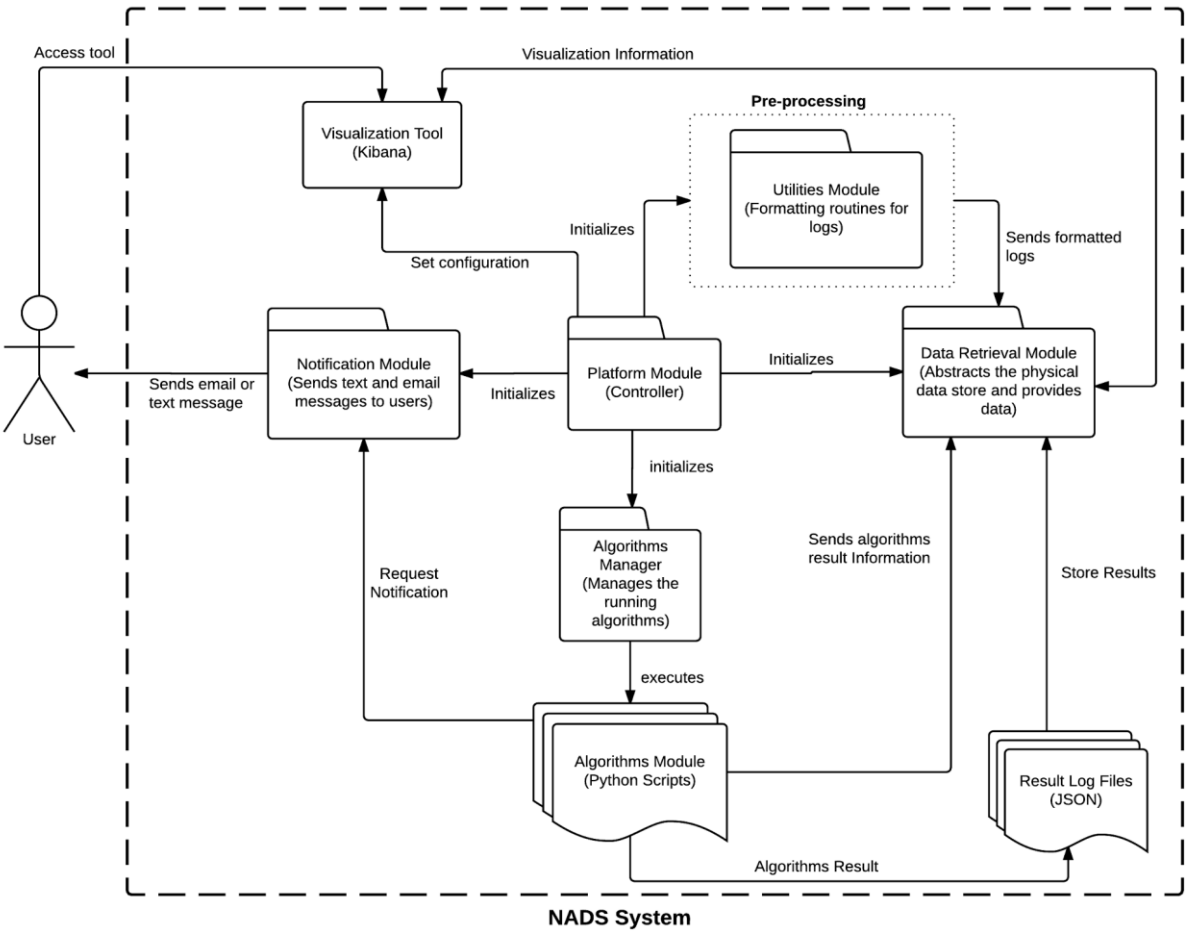


Figure 4. Block Diagram view of the entire system

The block diagram of the system shows how all of the modules are connected in the system. The Platform module manages all of the other modules. The Algorithms module in turn manages a set of algorithms that interact with the Notification module, the Data Retrieval module, and the visualization module. The data retrieval module interacts with the Utilities module to make sure that the log data is able to be ingested by the data retrieval system. The modules have the following interactions:

A. Platform Module to Notification Module
   - The Platform module supplies the Notification module its configuration options.
   - The Platform module also initialize the Notification module's thread of execution.
   - The Platform module sends a message to the execution thread of the Notification module to shut it down in the event of a system shutdown.

B. Platform to Visualization Module
   - The Platform module supply the Visualization module its configuration options.
   - The Platform module also initialize the Visualization modules thread of execution.

- The Platform module sends a message to the execution thread of the Visualization module to shut it down in the event of a system shutdown.

C. Platform to Utilities Module
   - The Platform module supplies the Utilities module its configuration options.
   - The Platform module also initialize the Utilities modules thread of execution.
   - The Platform module sends a message to the execution thread of the Utilities module to shut it down in the event of a system shutdown.

D. Platform to Data Retrieval Module
   - The Platform module supplies the Data Retrieval module its configuration options.
   - The Platform module also initialize the Data Retrieval modules thread of execution.
   - The Platform module sends a message to the execution thread of the Data Retrieval module to shut it down in the event of a system shutdown.

E. Platform to Algorithms Module
   - The Platform module supplies the Algorithms module its configuration options.
   - The Platform module also initialize the Algorithms modules thread of execution.
   - The Platform module sends a message to the execution thread of the Algorithms module to shut it down in the event of a system shutdown.

F. Utilities Module to Data Retrieval Module
   - Changes configuration file of logstash to contain desired regular expression for log input.

G. Algorithms Module to Individual Algorithms
   - Hands the individual algorithms the configuration options they need to execute
   - Performs the initial execution of each individual algorithm
   - Sends a shutdown message to individual algorithms in the case of system shutdown

H. Algorithms to Notification Module
   - Individual algorithms send a message to notify configured users of a specific anomaly
   - They can send a request for either text or email notifications or both

I. Algorithms to Data Retrieval Module
   - The algorithms send requests for data
   - The Data Retrieval module returns the data in the form of JSON objects
   - The algorithms send data of results to Data retrieval module.

J. Visualization Module to Data Retrieval
   - Queries for data that is need to visualize the result of an algorithm
   - Sends search queries to view data

Appendix F: Design Documentation

Platform Module Design

We will now describe the considerations that were taken into account when designing the NADS Platform Module.

Design Criteria, constraints and standards:

The NADS Platform module had to be developed as lightweight as possible in respect to the operations that it does. The module also had to be designed to be able to be run independent of the underlying operating system (OS). To handle these constraints, the system was developed in Java. All of the team members are familiar with Java, and as long as the Java Virtual Machine (JVM) is able to be run in the OS, the programs should execute successfully. Additionally, the platform class is just in charge of parsing the configuration file, and starting up the rest of the modules. Most of these operations are inexpensive as the work is deferred to a separate thread for each module. The most demanding operation that the platform does is parsing the configuration file and it is as simple as reading a JSON file and conveying the values to an appropriate place inside other objects.

Activity Diagram:

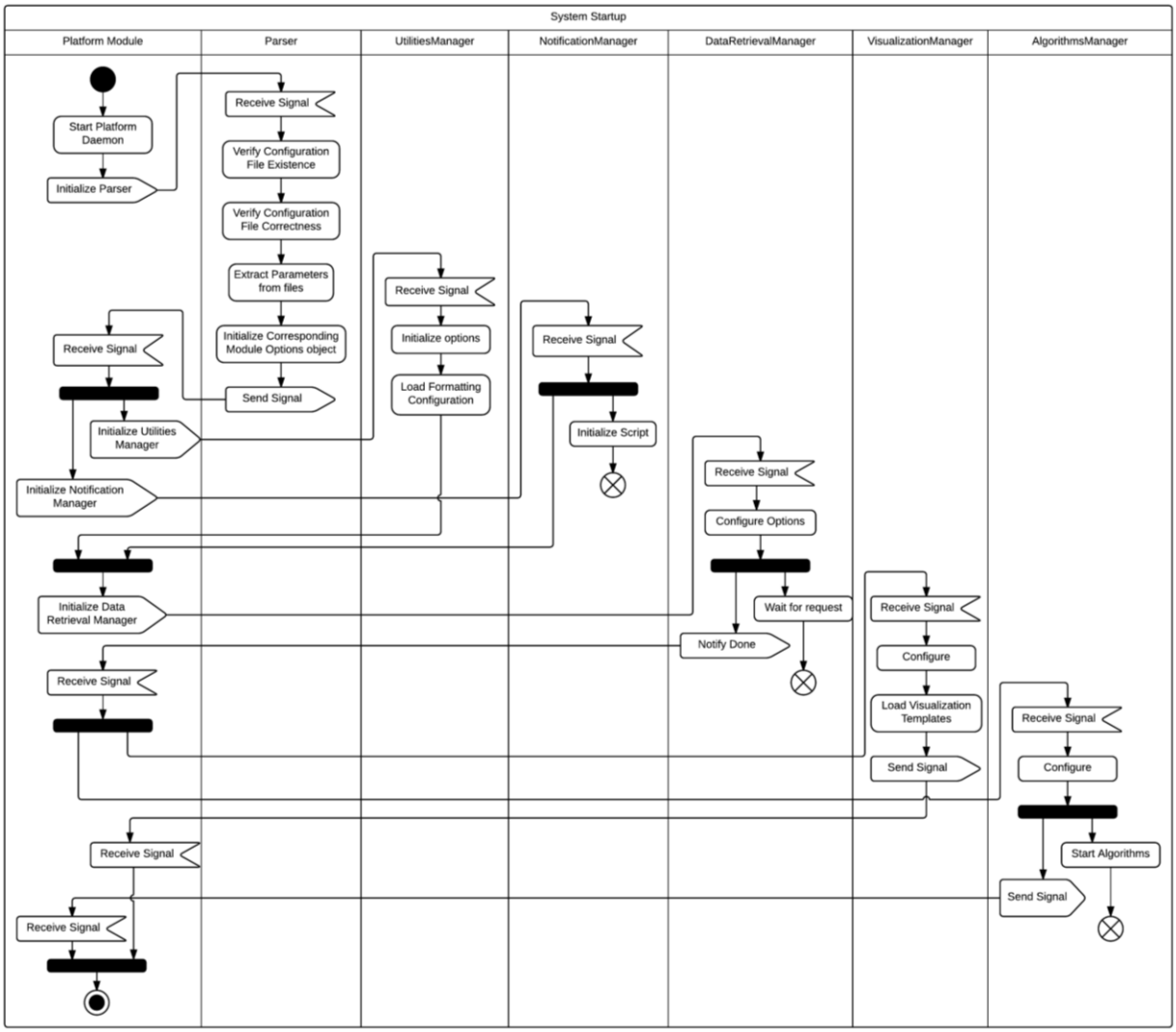


Figure 5: Activity Diagram of Platform Startup

Sequence Diagram:

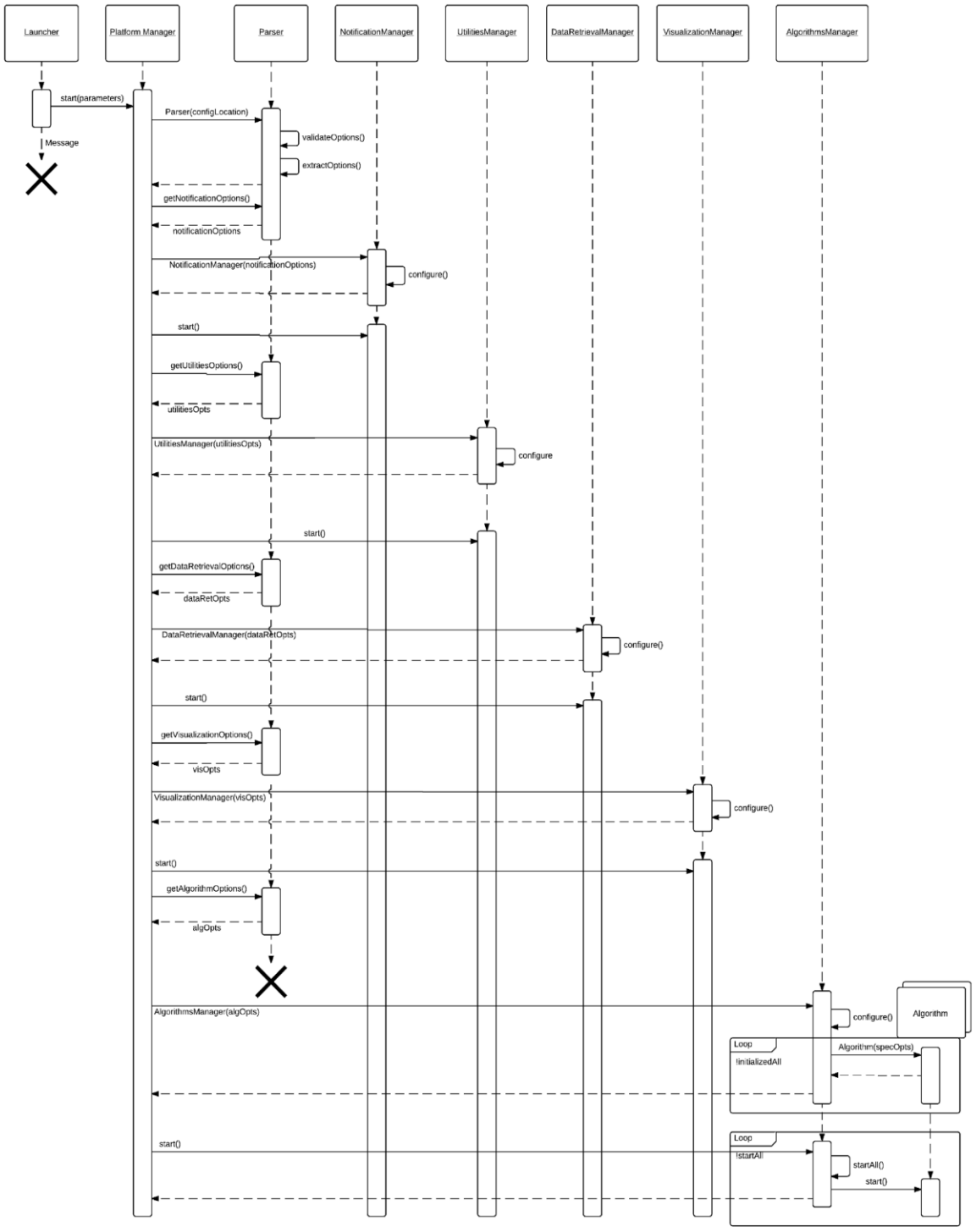


Figure 6: Activity Diagram of Platform Startup

Class Diagram:

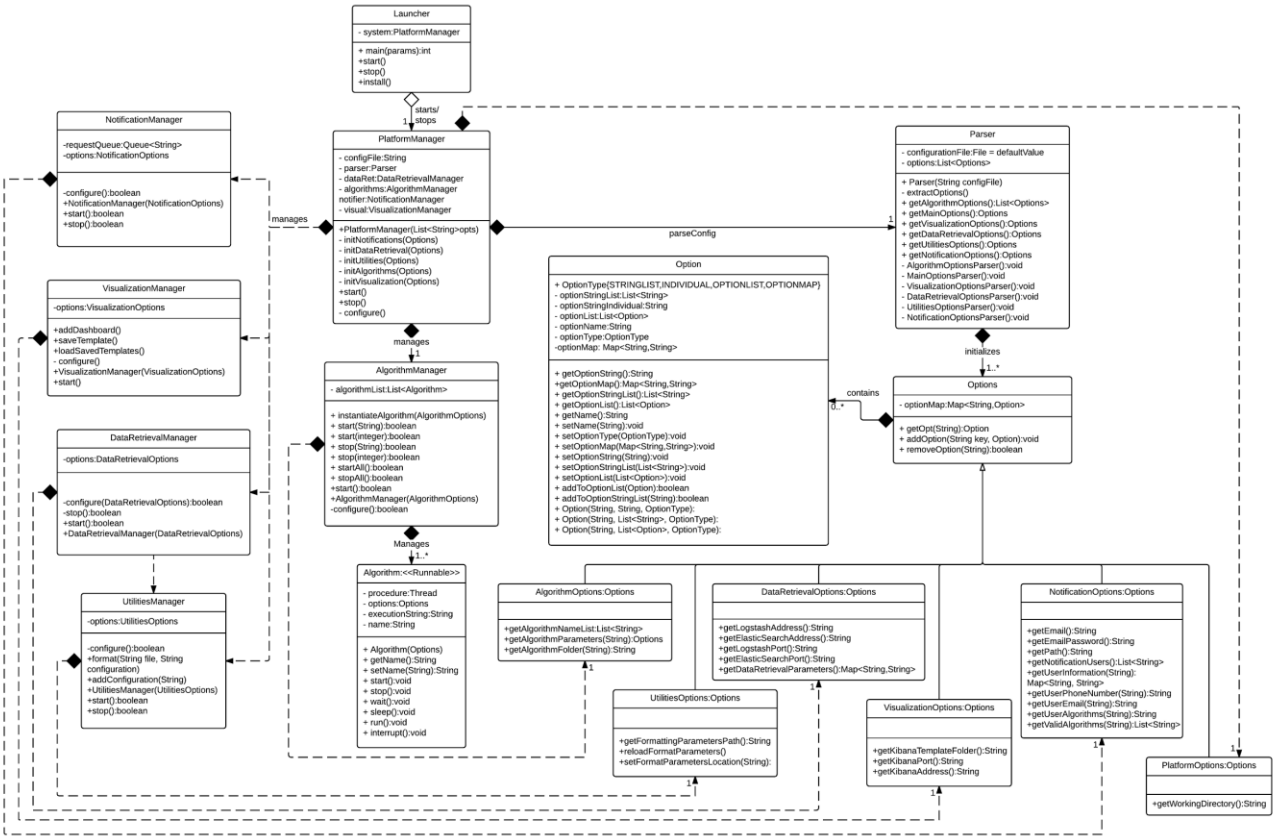


Figure 7: Class Diagram of Platform

Notification Design

The design considerations for the Notification module will now be explained.

Design Criteria, constraints and standards:

For the notification system our client needed a system that could send out email and text notifications as soon as one of the network anomalies was detected. As a general system constraint he specified that we couldn't use any paid software licenses. The team also set a constraint that the system should have as little dependencies as possible and be able to send the notifications to different users depending on the algorithms. The system would also have to be able to take in messages from the algorithms to send to the user.

Final Design Update:

The notification system was changed to follow a REST server/client model. The flowchart was incorporated partially in a server (which was implemented in Flask). The server just awaited a REST request to send out the notification.

Notification System Algorithm Flowchart:

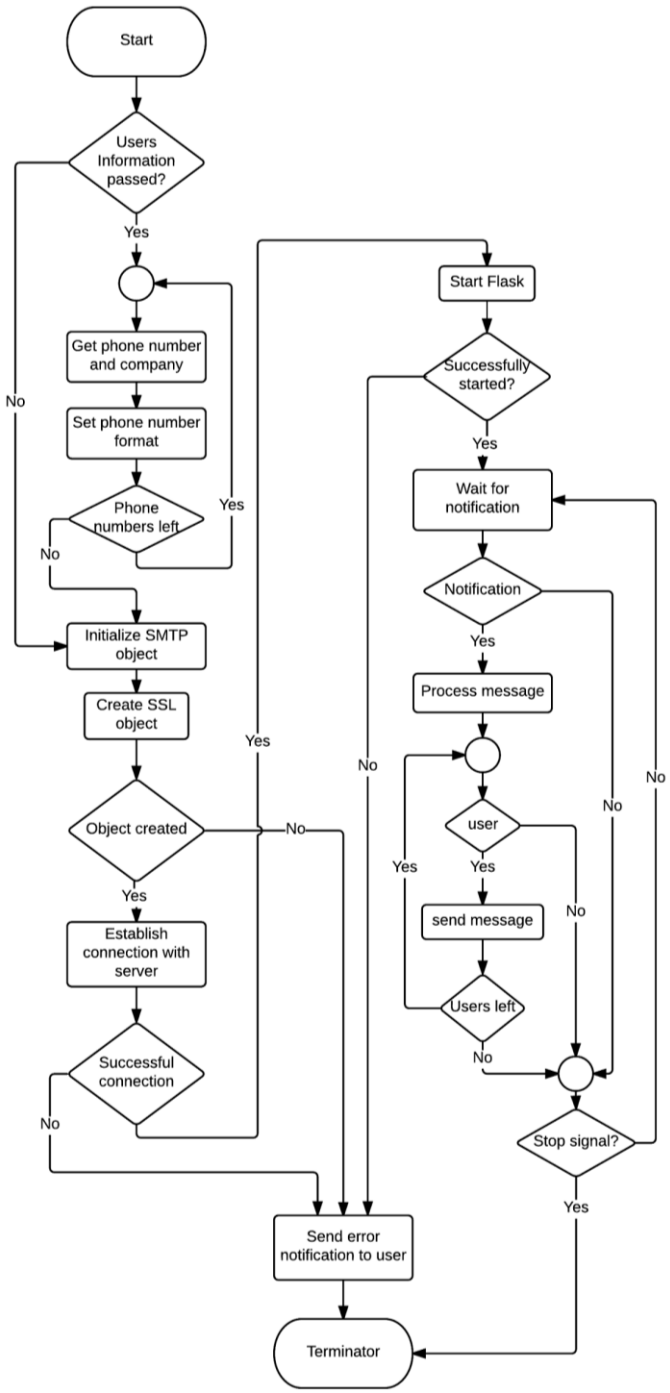


Figure 8: Notification System Flowchart

Data Retrieval Design

The design considerations for the Data Retrieval module will now be explained.

Design Criteria, constraints and standards:

For this design the only constraints we had from the client request was that we could retrieve the data needed without impacting the current configuration and that the solution be free of cost. The team however had the criteria that our data retrieval method be data store independent. This criteria was chosen so that various algorithms could be implemented with the same queries without having to worry about where the data came from or how it came.

Final Design Update:

As the team began to implement, some additional research produced that there is an existing elasticsearch plugin called Elasticsearch-SQL which essentially translates a SQL query into an Elasticsearch query. The team decided to modify this plugin to take a KQL query and translate it to SQL and then execute that over Elasticsearch.

System overview:

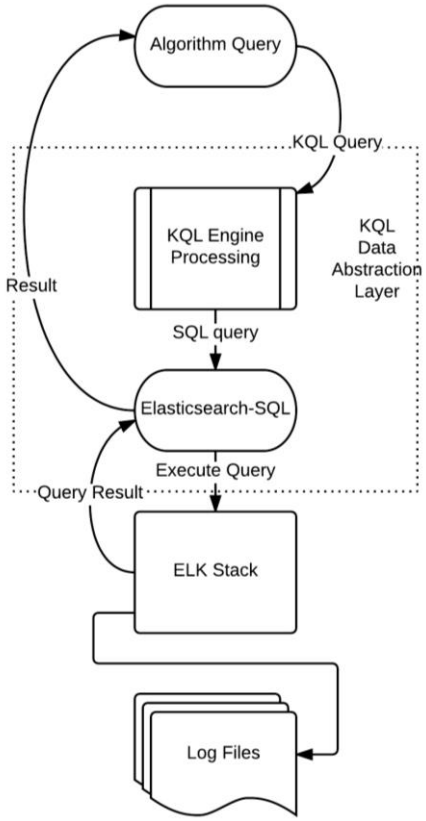


Figure 9: System architecture of Data Retrieval module

Visualization Design

The considerations taken into account when designing Visualization method to use will now be described.

Design Criteria, constraints and standards:

For this section we will talk about what the client wanted for the visualizations. We have to make the distinction though that this module was not design as we used an available visualization tool. This tool online required initial setup and configuration. The client desired a simple visualization screen that could give him situational awareness. The client desired to see the amount of protocol attacks, the frequencies of logins, and the dates of the attacks. He also desired for the data to be searchable through time periods.

For the Visualization the team opted to use Kibana. Through Kibana, the team can create dashboards which can be loaded into an ELK installation in order for the user to visualize some of the data. As the team did not build this tool, the team has only concentrated on the building of dashboards for it.

Prototype Dashboard:

Here is an example dashboard that was configured based on the requirements by the client:

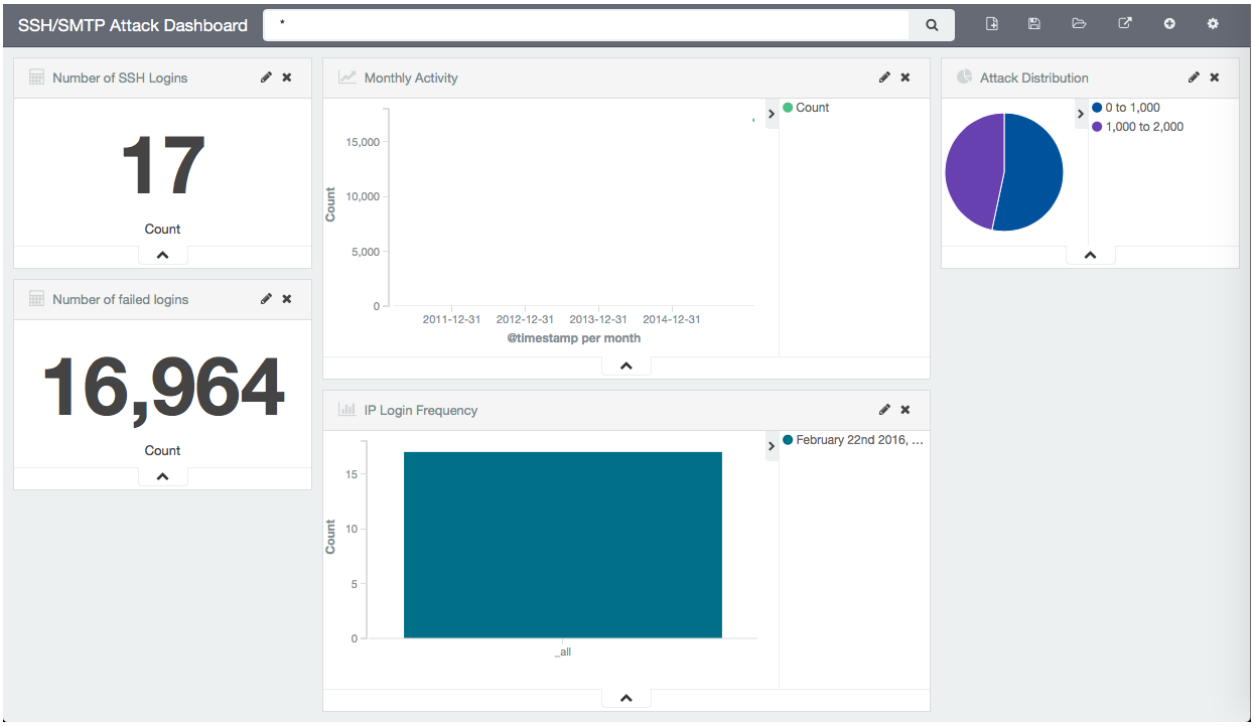


Figure 10: Prototype SSH/SMTP Visualization Dashboard

Prototype Dashboard Code:

Here is an example of the code needed to generate the prototype dashboard


```
{
   "_id": "SSH-slash-SMTP-Attack-Dashboard",
   "_type": "dashboard",
   "_source": {
     "title": "SSH/SMTP Attack Dashboard",
     "hits": 0,
     "description": "",
     "panelsJSON": "[{\"col\":1,\"id\":\"Number-of-SSH-
Logins\",\"panelIndex\":1,\"row\":1,\"size_x\":3,\"size_y\":2,\"type\":\"visualization\"},{\"col\":4
,\"id\":\"IP-Login-
Frequency\",\"panelIndex\":2,\"row\":4,\"size_x\":6,\"size_y\":3,\"type\":\"visualization\"},{\"c
ol\":10,\"id\":\"Attack-
Distribution\",\"panelIndex\":3,\"row\":1,\"size_x\":3,\"size_y\":2,\"type\":\"visualization\"},{\"c
ol\":1,\"id\":\"Number-of-failed-
logins\",\"panelIndex\":4,\"row\":3,\"size_x\":3,\"size_y\":2,\"type\":\"visualization\"},{\"id\":\"
Monthly-
Activity\",\"type\":\"visualization\",\"panelIndex\":5,\"size_x\":6,\"size_y\":3,\"col\":4,\"row\":1}
]",
     "optionsJSON": "{\"darkTheme\":false}",
     "uiStateJSON": "{\"P-5\":{\"spy\":{\"mode\":{\"name\":null,\"fill\":false}}}}",
     "version": 1,
     "timeRestore": false,
     "kibanaSavedObjectMeta": {
       "searchSourceJSON":
"{\"filter\":[{\"query\":{\"query_string\":{\"analyze_wildcard\":true,\"query\":\"*\"}}}]}"
     }
   }
 }
```

Final Dashboard:

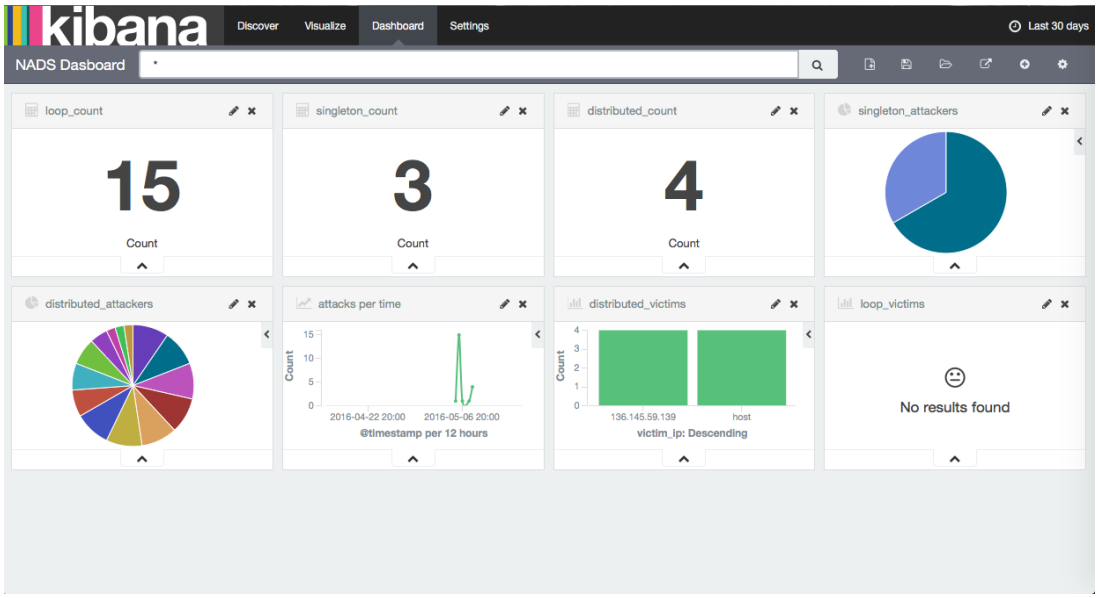


Figure 11: Visualization Dashboard

Algorithms Design

The considerations taken into account when designing the SSH and SMTP Attack detection algorithm will now be described.

SSH/SMTP Attack Detection

Design criteria, constraints and standards:

The algorithm for the SSH and SMTP detection has to correctly detect when a singleton and a distributed brute force attack is taking place within the department site. The algorithm also has to notify a user when this attack takes place with the type of the attack and the target of the attack.

Because the SSH and SMTP protocols both generate log when a login was successful or failed, the team decided to utilize the data found in the logs that the protocol generate and so the algorithm for SSH and SMTP detection will be the same, with different tuning parameters. The algorithm can be tailored, through a set of parameters, to a specific site login pattern, speed, and detection accuracy. These parameters have to be determined from an analysis of previous site login data, and client notification expectations. Since the algorithm has to detect distributed attacks, the algorithm itself has to monitor various nodes. For this, a client-server model was used. A centralized server will process all side wide requests to perform analysis and notify users when an attack is detected.

The team opted to use the Log-Based distributed attack algorithm [16]. The algorithm consists of monitoring changes to the mean distribution of the amount

of login failures to the site. It utilizes a Cumulative sum approach to monitor this change. Whenever the cumulative sum passes a threshold, events (groups of logins) are packaged into an epoch and sent to analysis in a classifier.

Final Design Updates:

The algorithm used was the same one as the paper [REFERENCE]. The algorithm was modified however to perform an additional analysis every time the modulus of the out of control events and the out of control average run length equaled zero. This was added to stop the attack as soon as possible. Since the purpose of the paper was to detect when the attacks started/stopped, we added this feature in to prevent the continuation of an attack/end it as early as possible.

An additional modification that was done was to automate the generation of the graphs and the extraction of the attack clusters. In essence the algorithm generates the bipartite graph and ties the Clients with a server. We would then perform cluster extraction of this resulting graph. This was done to eliminate the need for manual input on the algorithm.

Another modification done was the integration of an IP blocking system to block IPs from monitored systems. This was done using the python-iptables package and a rest api. A call gets sent to the monitored nodes with the ip that is desired to be blocked.

Original Class Diagram:

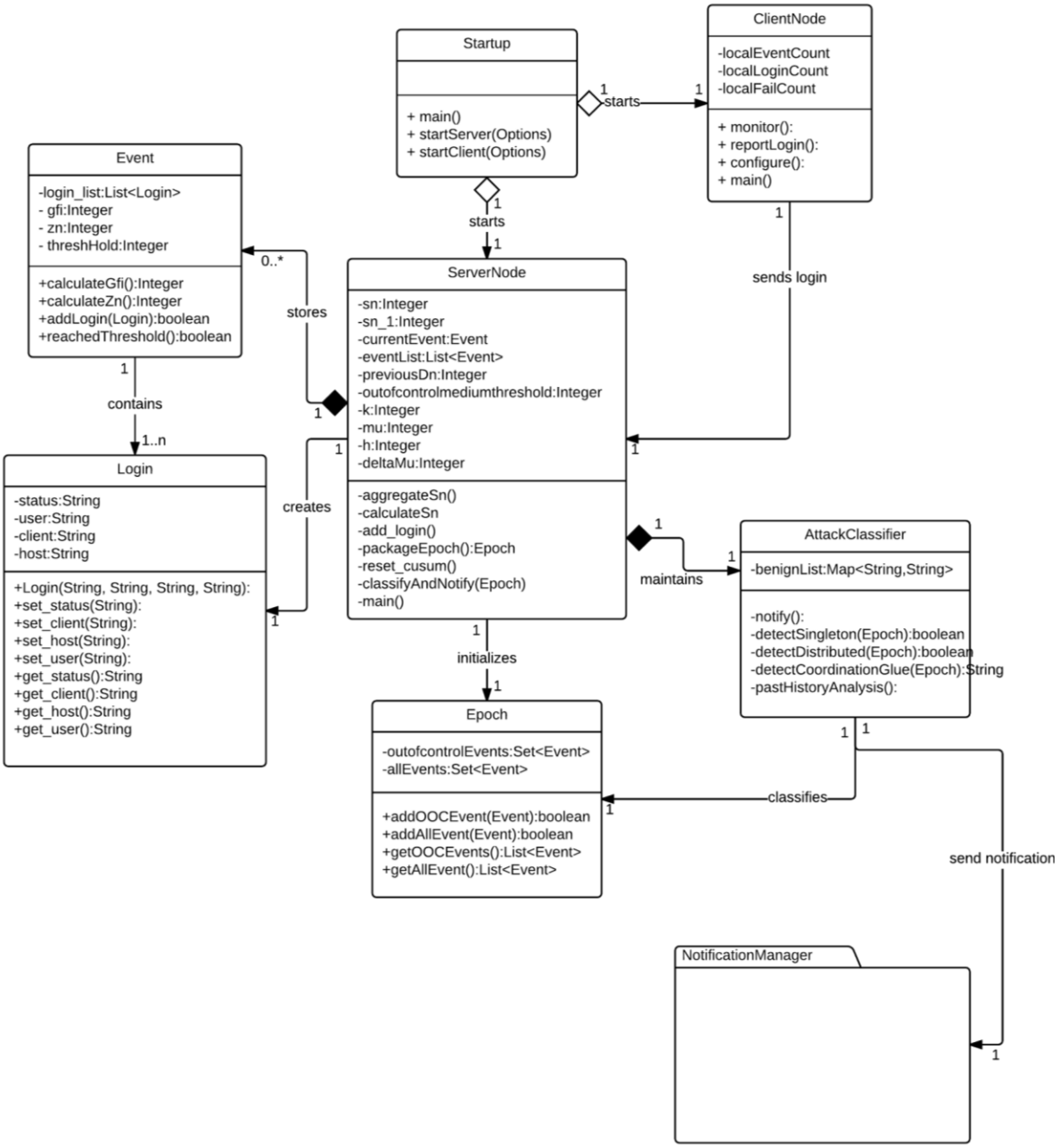


Figure 12: SSH/SMTP detection class diagram

Updated Class Diagram:

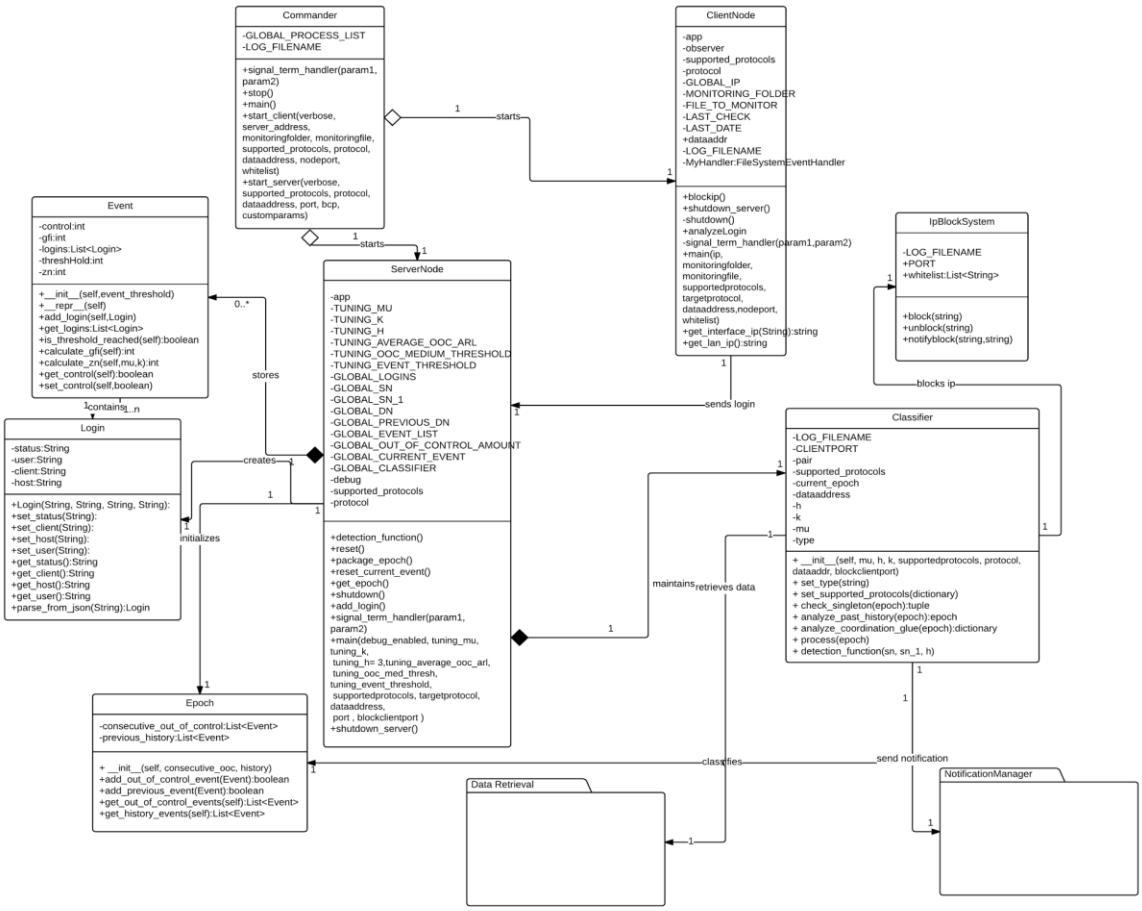


Figure 13: Updated SSH/SMTP Protocol attack detection class diagram

Activity Diagram:

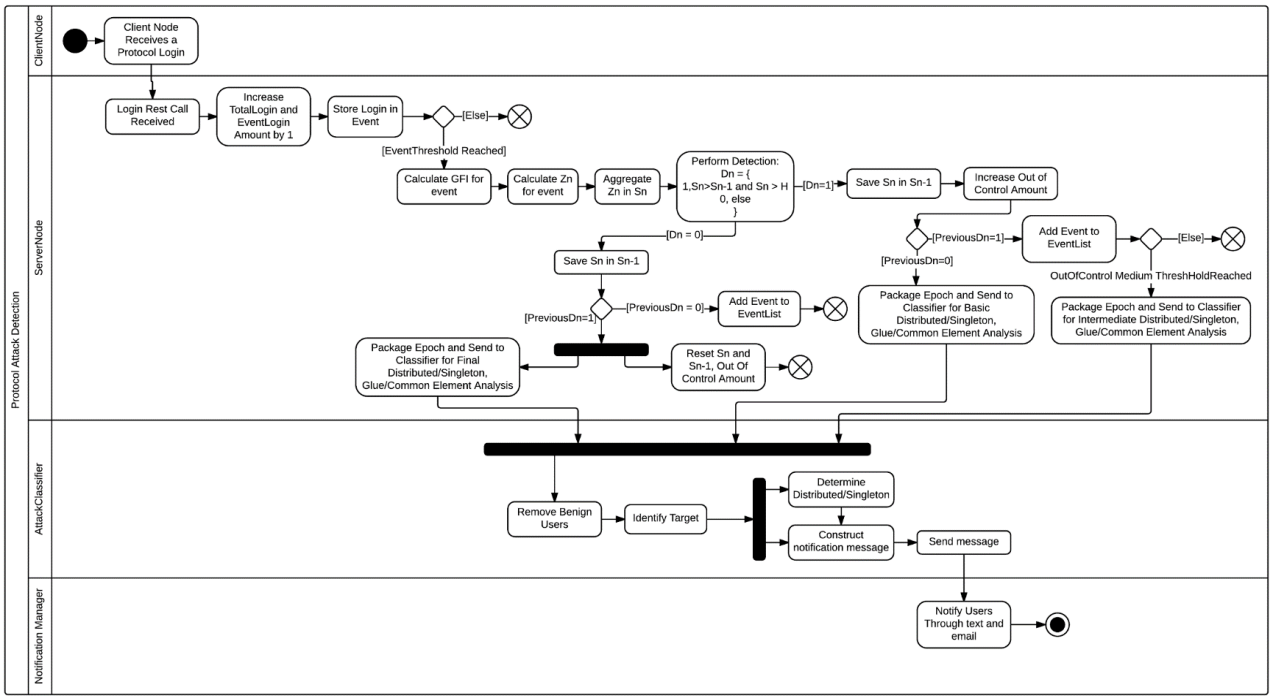


Figure 14: SSH/SMTP detection activity diagram

Loop Detection Algorithm

The considerations taken into account when designing Loop detection algorithm will now be described.

Design criteria, constraints and standards:

For the loop detection algorithm it was desired that the algorithm detected the loop as soon as possible. It was also desired that switches that would be monitored could be added or removed easily. The algorithm would also notify the system administrator as soon as the detection happened with the IP address of the switch as well as the interface that was blocked.

In order to do this the team choose and active querying system. The system constantly queries the switches in the network to see the STP state of each interface. If the scripts detects that an interface is being blocked by STP an alert is sent to the user with the necessary information. The system know which switches to check through a pre-configured list. Implementing the system this way allowed us a way to detect the loop in a way that met all of our criteria.

Final Design Updates:

The algorithm that was used at the end was the same one mentioned above. Although this algorithm was the one that was implemented at the end this was not always the case. Originally the algorithm was based on a callback system on

which each switch would be set up with SNMP trap that would call back to our script. The diagrams of this method can be seen below. This method however was discarded due to the fact that switches used in the school did not seem to allow custom traps to be created. Once this issue was found the team designed and implemented the new algorithm.

Network Switch Loop Detection Algorithms Flowchart:

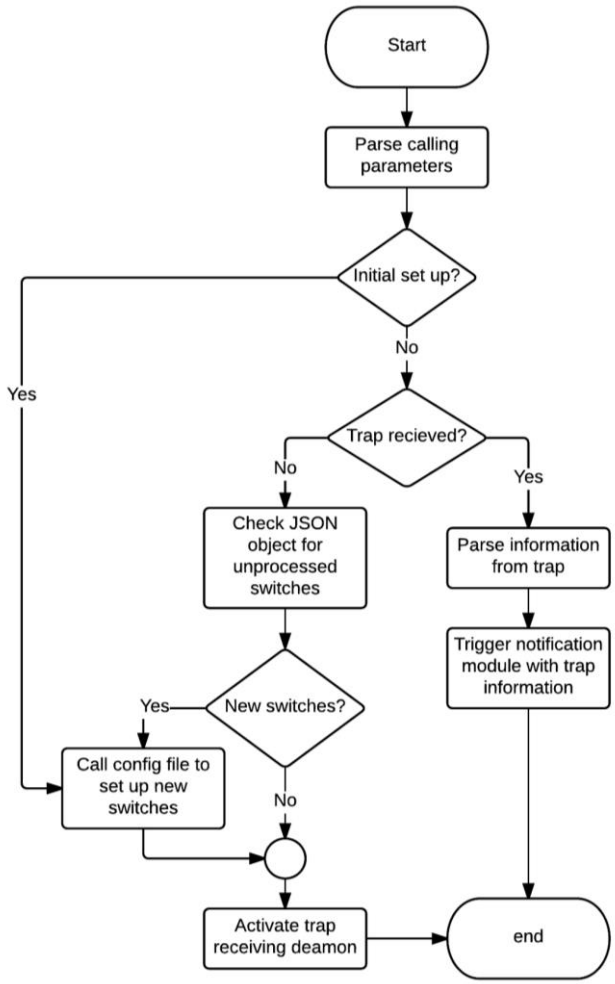


Figure 15: Old Network Switch Loop Detection Flowchart (a)

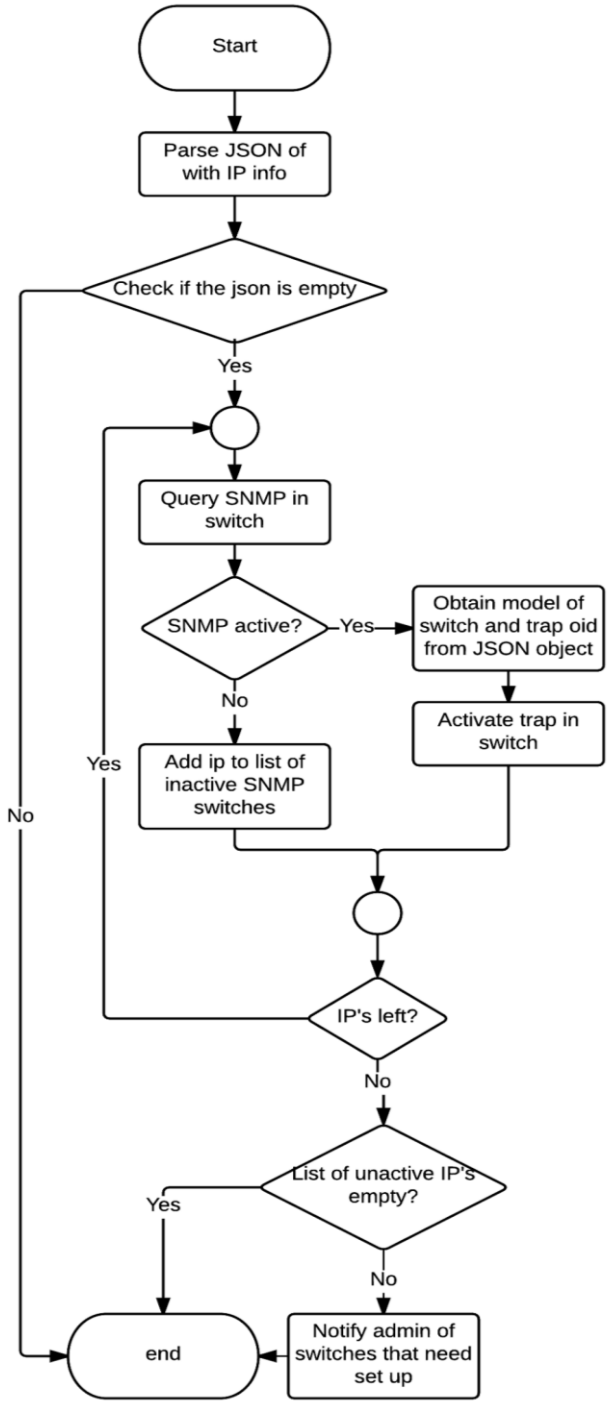


Figure 16: Old Network Switch Loop Detection Flowchart (b)

Activity diagram:

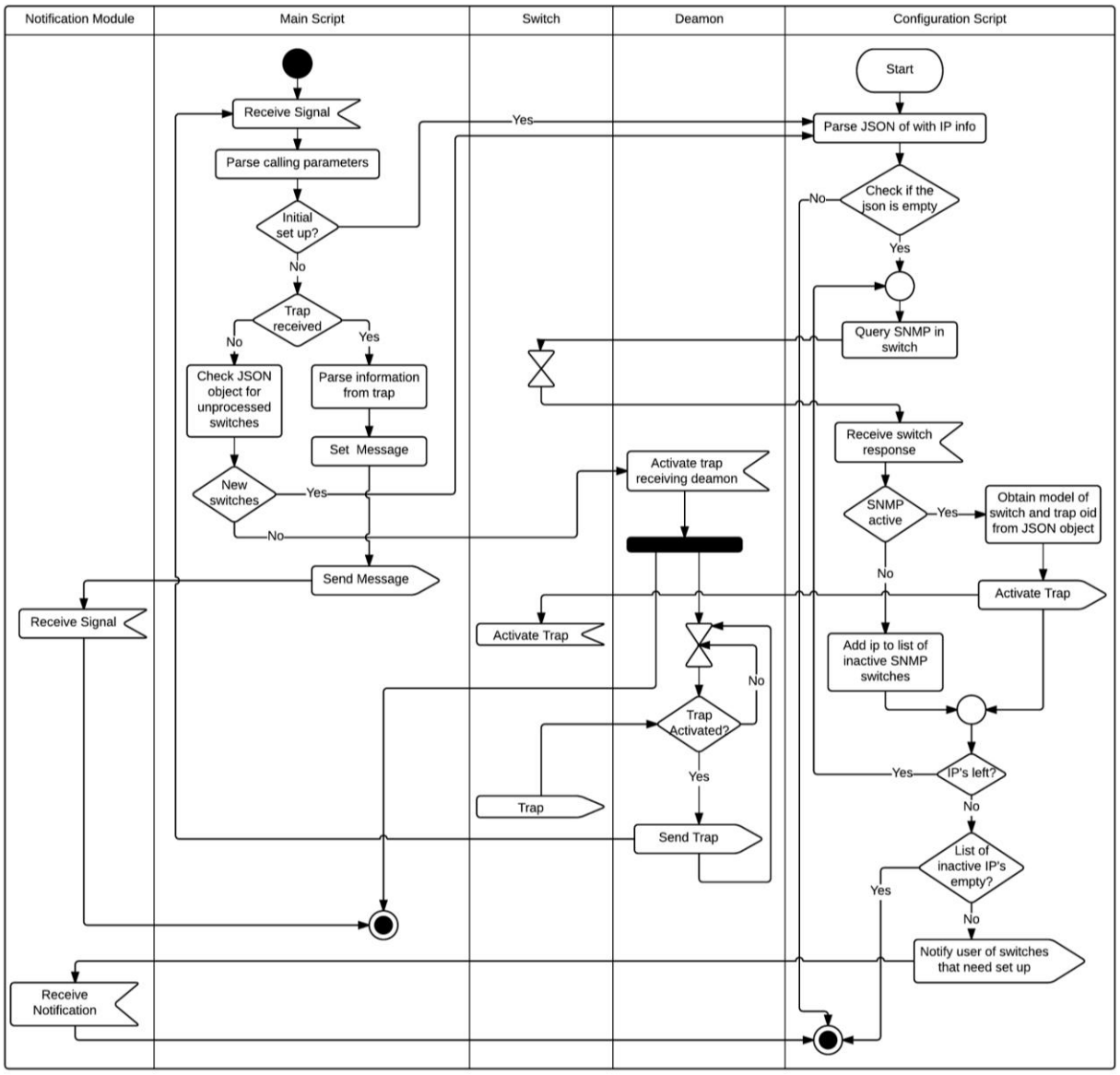


Figure 17: Old Loop Detection Activity diagram

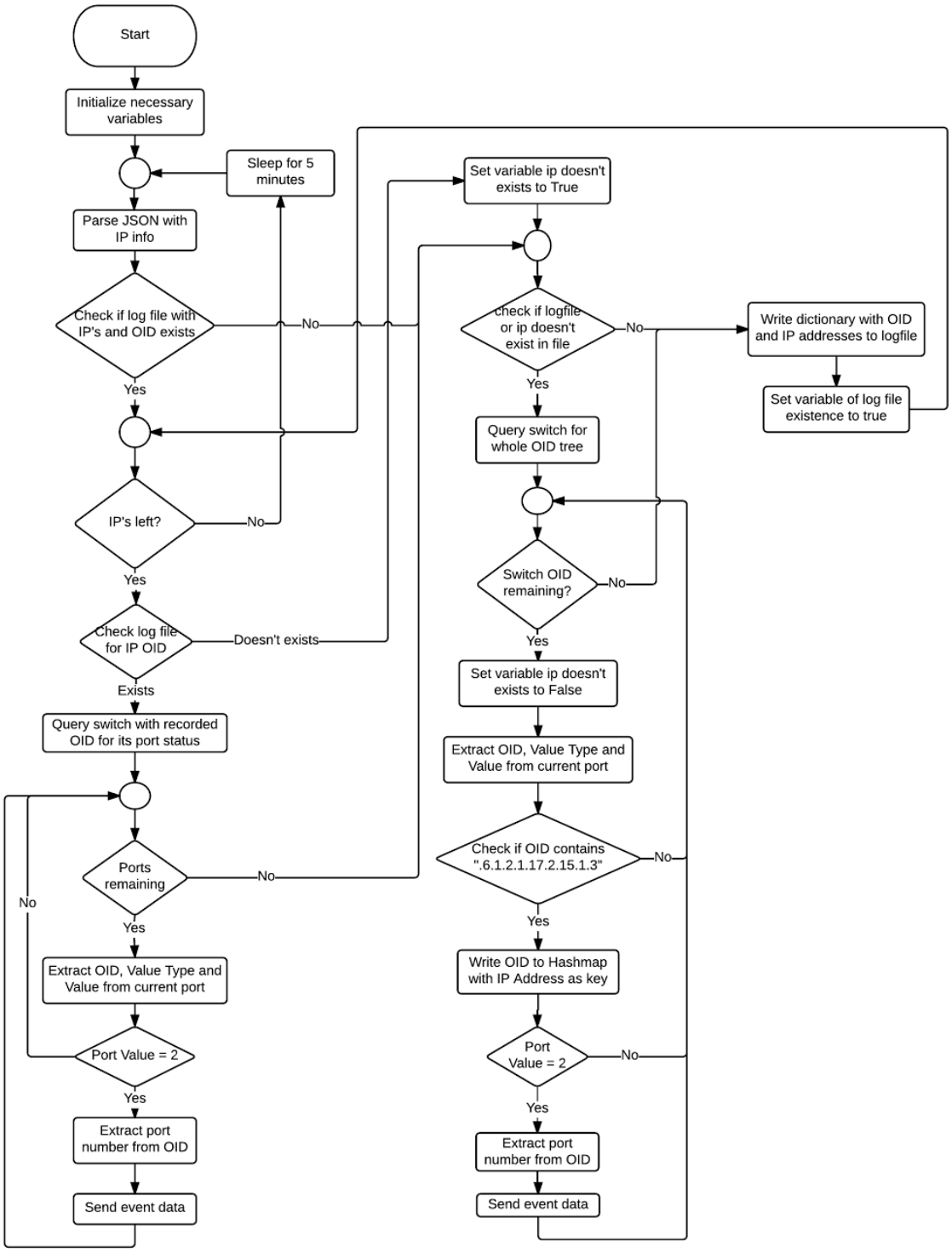


Figure 18: Network Switch Loop Detection Flowchart

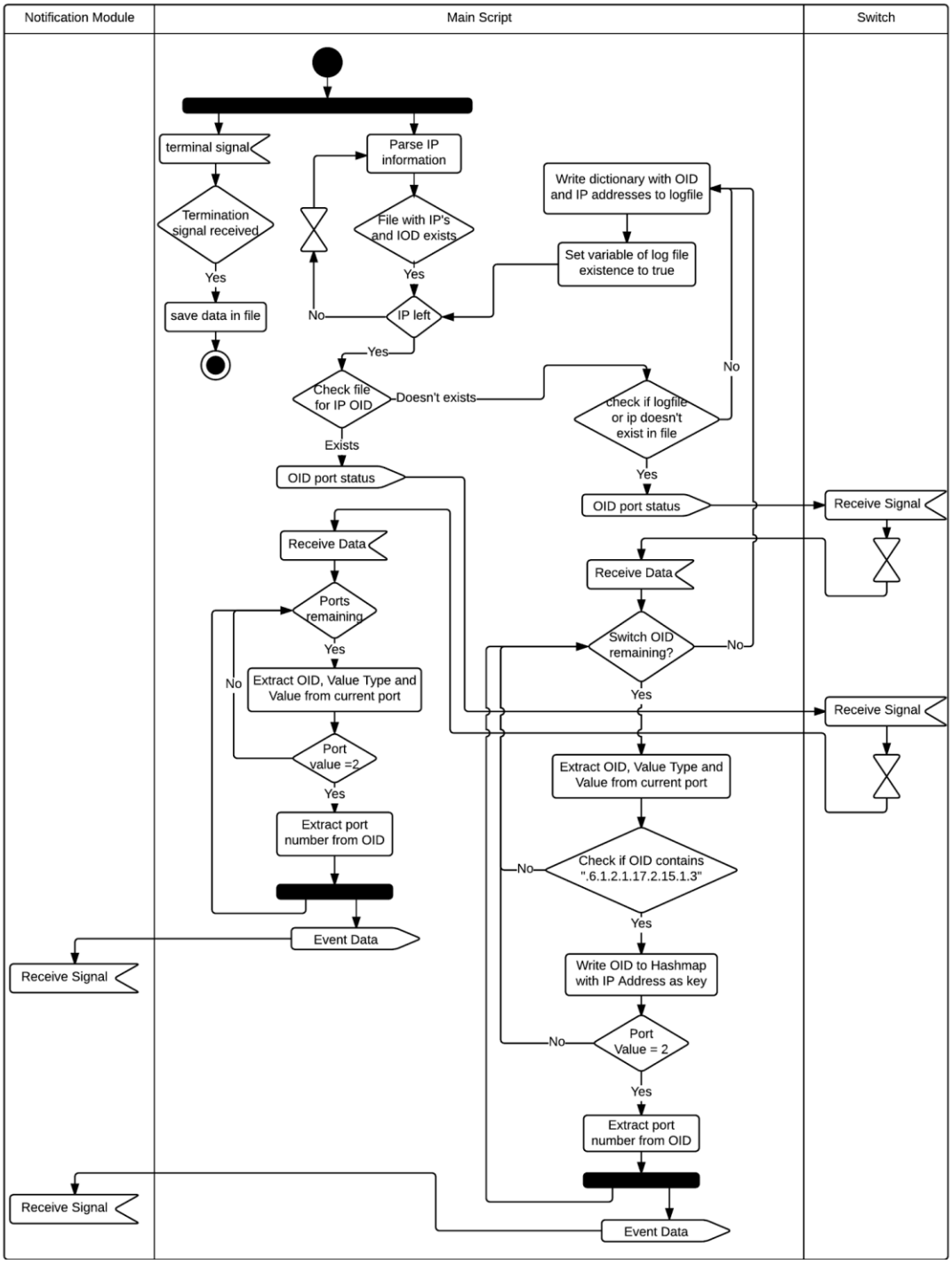


Figure 19: Loop Detection Activity diagram

Utilities Design

The considerations taken into account when designing the Utilities Module will now be described.

Design criteria, constraints and standards:

The mechanism used to format the logs was a regular expression in Grok. This regular expression breaks a log message into the fields that are used to index and search the data. The regular expression had to have support for two distinct log patterns.

Regular Expression:

%{SYSLOGBASE} %{WORD:nothing} %{WORD:nothing2} %{WORD:nothing3} %{WORD:user} %{WORD:nothing4} %{IP:user_ip} %{WORD:nothing5} %{NUMBER:port} %{WORD:user_deamon}

Appendix G: Testing Plan

The team decided to perform a testing plan based on modules. The tests verify reliability and correctness of the software. Most of the tests involve unit tests and simulations of situations that the team deemed appropriate to verify the behavior of the platform.

Testing Tools

The following is a listing of the required software for the tests that NADS used.

- JUnit
  - A Java framework that lets users write repeatable tests.
  - Most of the module tests were written in JUnit as the majority of the modules have a major Java component.
  - JUnit tests were automated with the Maven Surefire plugin.
- Maven SureFire
  - A Maven build plugin that automatically executes unit tests within a specified location and returns their results.
  - A standardized way to run tests and assure platform build correctness
- PyUnit
  - A Python framework that lets users write repeatable tests.
  - It is the equivalent of JUnit for Python.
  - The framework was utilized to verify the correctness and reliability of the Anomaly Detection algorithms and Notification Module as they were written in Python.

Testing Characteristics, Procedures, Expected Results and Actual Results

Individual Modules

Platform Module

- Configuration Parsing
  - Description
    - The team tested whether a configuration file is able to be parsed successfully into the corresponding 6 types of options (i.e. Notification Options)
  - Procedures
    - 6 unit tests verify that each of the 6 specific module options objects (Platform, notification, algorithm, etc.) are initialized successfully and some dummy data is able to be retrieved from them.
    - 1 unit test was implemented to verify the case that a configuration file has a parsing error (i.e. is not correctly written as JSON)

        - 1 unit test was implemented to verify incomplete configurations (i.e. algorithm is missing a default entry such as algorithm name)
        - 1 unit test verify system stability in case there is no configuration file given or is not present
    - Expected Results
        - The 6 initial unit tests should always pass (there should always be some default options that was always be present for each module). If this test fails, the parsing system will need to be revised to ensure that data is being assigned to the correct properties of each object
        - The following unit tests (which verifies correct format) should always fail with correctly written configuration files. It should always succeed if there is a JSON parsing error. This test is to ensure reliability with the parsing of the configuration file.
        - The incomplete configuration file unit test should once again always fail if a configuration file is correctly written. It should succeed in the case that a required element is present in the configuration file. It will also fail in the case that the configuration file is not able to be parsed correctly.
    - Actual Results
        - All unit tests passed as all 6 modules were able to extract their options. The parsing unit was also able to detect the errors on the configuration file and react accordingly.

- Modules Initialization
    - Description
        - The team tested whether a module is able to complete an initialization. This consists of retrieving information from its options object, and correctly configuring itself (storing and interpreting the information) without any errors.
    - Procedures
        - The team developed a total of 6 unit tests (two for each module) verifying that a module is able to start correctly (finish its configuration routine) or notify the system of an error.
    - Expected Results
        - 6 unit tests are expected to pass whenever a correct configuration file is provided to the platform. If these tests fail, the specific (module dependent) functionality of the configuration objects must be revised to verify correctness.
    - Actual Results
        - All the modules were able to start correctly with their respective configurations as such all test were passed.

- Service Stop
  - Description
    - The team tested for a graceful shutdown whenever the service is required to be stopped.
  - Procedures
    - The team wrote a unit test that verifies log files when a stop signal is sent to the platform service. This should send out an adequate message to each module and they must produce a log file that indicates their termination.
    - The team also tested this behavior using a manual linux command such as: “sudo service *servicename* stop”.
  - Expected Results
    - The test is expected to pass if the system message handling is implemented correctly. If it fails, then the team must revise these routines to make sure a shutdown message is received correctly and passed to the individual modules.
  - Actual Results
    - The service was able to shut down gracefully.

- Service Start
  - Description
    - The team tested for a graceful start whenever the service is required to be started either manually through a system command or programmatically.
  - Procedures
    - The team wrote a unit test that verifies log files when a start signal is sent to the platform service. This sends out an adequate message to each module and they must produce a log file that indicates their successful start.
    - The team also tested this behavior using a manual linux command such as: “sudo service *servicename* start”.
  - Expected Results
    - The test is expected to pass if the system message handling is implemented correctly. If it fails, then the team must revise these routines to make sure a start message is received correctly and passed to the individual modules.
  - Actual Results
    - The service once installed was able to be started without problems.

Notification Module

- Notify all through email
  - Description
    - The team tested the successful and unsuccessful sending of an email from the Notification module to configured users.

    - Procedures
        - The team wrote a unit tests. This performed a test to send a sample email to configured addresses and send an email to an incomplete address to see how the module responds to an error.
    - Expected Results
        - The test is expected to pass if the system has been configured to send emails appropriately and if the system is able to handle an error appropriately and recover. If it fails, then the team must revise the email configuration and the emailing service.
    - Actual Results
        - The module was able to send an email correctly as well as being able to detect when an incorrect email was inputted.

- Notify all users through text
    - Description
        - The team tested the successful and unsuccessful sending of a text message from the Notification module to configured users.
    - Procedures
        - The team wrote a unit tests. This performed a test to send a sample text message to configured numbers and a text message to an incomplete number to see how the module responds to an error.
    - Expected Results
        - The test is expected to pass if the system has been configured to send text messages appropriately and if the system is able to handle an error when sending a text message and recover. If it fails, then the team must revise the text message configuration and the text message service.
    - Actual Results
        - The module was able to send an text message correctly as well as being able to detect when an incorrect number was inputted.

Algorithm Module

- Algorithm Initialization
    - Description
        - The team tested the ability of the module to successfully launch and stop an algorithm.
    - Procedures
        - The team wrote a unit test for each of the methods that are called to start and stop algorithms.
    - Expected Results
        - The tests are expected to pass if an algorithm object has correctly implemented the methods. If not, the team has to revise the lifecycle methods of the algorithm class.

- Actual Results
  - The module was able to correctly start and gracefully stop an algorithms script.

- SSH/SMTP Detection
  - Login detection
    - Description
      - The team tested the ability of a node to be able to process and notify whenever a login attempt is made to the node.
    - Procedures
      - The team wrote a unit test that verifies if the central node is sent a POST call when a remote user logs through a protocol unto a node.
    - Expected Results
      - The test is expected to pass if the monitoring routine is correctly implemented in a client node. If the test fails the team must revise the monitoring strategy.
    - Actual Results
      - The script was able to detect and make a post call when a user log in through a protocol.

  - Event detection
    - Description
      - The team tested the ability of the module to successfully generate an event object when an event threshold is reached.
    - Procedures
      - The team wrote a unit test that verifies an event object generation when a certain request limit is reached.
    - Expected Results
      - The tests are expected to pass if the team has correctly initialized the event threshold limit and the event threshold verification method.
    - Actual Results
      - The script was able to generate and event object at the specified threshold limit.

  - Epoch detection
    - Description
      - The team tested the ability of the module to successfully package and send an attack epoch object.
    - Procedures
      - The team wrote a unit test that simulates an attack to the protocol.

- Expected Results
  - The test is expected to succeed if the team has correctly implemented the detector function and some adequate parameters for the detection algorithm. If not, the team must revise the algorithm parameters and the implementation of the detection function.
- Actual Results
  - The script was able to send an epoch object upon detection was an attack was started.

- Distributed Classification
  - Description
    - The team tested the ability of the module to successfully identify and classify a distributed attack.
  - Procedures
    - The team wrote a unit test that simulates a distributed attack. The test simulates more than 10 different remote hosts attempting to break into the site. The attack varies slightly from the normal login behavior of the school, yet it have a variety of remote hosts to make it stealthy and distributed.
  - Expected Results
    - The tests are expected to pass if the detection algorithm parameters are set up correctly (i.e. the normal login activity is correctly configured, the thresholds are set accordingly). If the test fails, the team must revise the algorithm parameters and recalculate the normal login activity distribution.
  - Actual Results
    - The system was able to detect and classify an attack as distributed.

- Singleton Classification
  - Description
    - The team tested the ability of the module to successfully identify and classify a singleton (1 user) bruteforce attack.
  - Procedures
    - The team wrote a unit test that simulates a singleton attack. The test have one user repeatedly try to login into the site.
  - Expected Results
    - The tests are expected to pass if the detection algorithm parameters are set up correctly (i.e. the normal login activity is correctly configured, the thresholds are set accordingly). If the test fails, the team must revise the algorithm parameters and recalculate the normal login activity distribution.

        - Actual Results
            - The system was able to detect and classify an attack as a singleton.

    - Attack Notification
        - Description
            - The team tested the ability of the module to successfully notify a user whenever an attack is discovered.
        - Procedures
            - The team wrote unit tests to simulate an attack and corroborate whether the identification of the attack is successful and a consequent notification request is sent out.
        - Expected Results
            - The tests are expected to pass if the detection algorithm parameters are set up correctly (i.e. the normal login activity is correctly configured, the thresholds are set accordingly). If the test fails, the team must revise the algorithm parameters and recalculate the normal login activity distribution. The team must also verify whether the Notification module is correctly set up and implemented.
        - Actual Results
            - The system was able to notify a user as soon as it detected an attack.

Loop Detection

- SNMP Connection
    - Description
        - The team tested the ability of the module to successfully connect to a network switch through SNMP
    - Procedures
        - The team wrote a unit test that tries to establish a connection to one of the configured routers.
    - Expected Results
        - The test is expected to complete successfully if the team has incorporated correctly the SNMP API they have looked into. If the test fails, the team must see if it is a configuration issue or look into another framework.
    - Actual Results
        - The script was able to connect and query a switch through SNMP.

- Loop Simulation, Detection, and Notification
    - Description
        - The team tested the ability of the module to successfully notify administrators of the presence of a network loop.

- Procedures
  - The team wrote a unit to verify that a loop is happening through a query to the switch using snmp. The test also notifies the user of the loop detected. The team simulated a network switch loop for the test to be executed.
- Expected Results
  - The test is expected to complete successfully if the querying and parsing mechanisms were correctly implemented. It will also complete successfully if the Notification module was implemented adequately.
- Actual Results
  - The scripts was able to detect a loop whenever it occurred as well as notify the user of the ip and interface on which the loop occurred.

Data Retrieval

- Send Data
  - Description
    - The team tested the ability of the module to generate a file for the logstash module through a post call.
  - Procedures
    - The team wrote a unit test to simulate a call to the data retrieval modules and compared the outputted file to the data that sent..
  - Expected Results
    - The test is expected to pass if the data in the files is the same as the data that was sent. The test is expected to pass if the simple java HTTP server was correctly implemented.
  - Actual Results
    - The module was able to correctly output the data to the file.

- Get Data
  - Description
    - The team tested the ability of the module to successfully retrieve data according to the ontology that is being used to abstract the data store.
  - Procedures
    - The team wrote a unit tests to perform a simple query through a post call and verify that the query is returned in an expected format.
  - Expected Results
    - The test is expected to pass if the log ingestion backend mechanism has been correctly set up and if the ontology has been correctly implemented as well as if the simple java HTTP server was correctly implemented..
  - Actual Results
    - The post call returned the correct data store in the back end.

- Simple *select from where* query
  - Description
    - The team tested the ability of the module to successfully execute a *select from where* query.
  - Procedures
    - The team wrote a unit test to input a *select from where* query and verify that it is translated and executed correctly to the log retrieval mechanism.
  - Expected Results
    - The test is expected to be successful if the query is analyzed correctly and the fields are correctly placed in the search parameters of the backend search solution. It is also expected that the search results are returned and processed adequately.
  - Actual Results
    - A simple select from query was conducted and returned the correct data from the ELK stack.

Integration Tests

Platform and notifications integration

- Startup
  - Description
    - The team tested the ability of the module correctly start after it has been integrated with the platform.
  - Procedures
    - The team wrote a unit test to pass some configuration parameters from the platform to the notification system and verify that it initializes completely
  - Expected Results
    - The test is expected to be successful if both modules were implemented adequately.
  - Actual Results
    - The test passed completely as both modules were able to start without problems.

- Stop
  - Description
    - The team tested the ability of the module correctly stop after it has been integrated with the platform.
  - Procedures
    - The team wrote a unit test to send a stop signal from the platform to the manager.

    - Expected Results
        - The test is expected to be successful if both modules were implemented adequately. If not the team must review the shutdown routine in the notification manager.
    - Actual Results
        - The test passed as both modules were able to shut down gracefully.

- Send both notification types
    - Description
        - The team tested the ability of the module correctly send a text and email message after it has been integrated with the platform.
    - Procedures
        - The team wrote a unit test to send a signal to notify users through text and email after the notification module is started from the platform.
    - Expected Results
        - The test is expected to be successful if both modules were implemented adequately. If not the team must review the routine to receive messages and send them.
    - Actual Results
        - The module was able to successfully send both a text and email message.

Platform and data retrieval

- Startup
    - Description
        - The team tested the ability of the module correctly start after it has been integrated with the platform.
    - Procedures
        - The team wrote a unit test to pass some configuration parameters from the platform to the data retrieval system and verify that it initializes completely.
    - Expected Results
        - The test is expected to be successful if both modules were implemented adequately.
    - Actual Results
        - The test passed completely as both modules were able to start without problem

- Stop
    - Description
        - The team tested the ability of the module correctly stopping after it has been integrated with the platform.

- Procedures
  - The team wrote a unit test to send a stop signal from the platform to the manager.
- Expected Results
  - The test is expected to be successful if both modules were implemented adequately. If not the team must review the shutdown routine in the data retrieval manager.
- Actual Results
  - The test passed as both modules were able to shut down gracefully.

Simple query

- Description
  - The team tested the ability of the module correctly return information from a select from where query.
- Procedures
  - The team wrote a unit test to send a query from a remote module to the data retrieval and see if it serves the request.
- Expected Results
  - The test is expected to be successful if both modules were implemented adequately.
- Actual Results
  - A simple query was conducted and returned the correct data from the ELK stack.

Platform and algorithms

- Startup
  - Description
    - The team tested the ability of the module correctly start after it has been integrated with the platform.
  - Procedures
    - The team wrote a unit test to pass some configuration parameters from the platform to the algorithms system and verify that it initializes completely.
  - Expected Results
    - The test is expected to be successful if both modules were implemented adequately.
  - Actual Results
    - The test passed completely as both modules were able to start without problem

- Stop
  - Description
    - The team tested the ability of the module correctly stopping after it has been integrated with the platform.

    - Procedures
        - The team wrote a unit test to send a stop signal from the platform to the manager.
    - Expected Results
        - The test is expected to be successful if both modules were implemented adequately. If not the team must review the shutdown routine in the algorithms manager.
    - Actual Results
        - The test passed as both modules were able to shut down gracefully.

- Event Simulations
    - Description
        - The team tested the events that were described in the individual algorithms, but utilizing the actual notification manager and data retrieval module.
    - Procedures
        - The team wrote unit tests to retest the proposed anomalies.
    - Expected Result
        - The test is expected to be successful all the previous modules were implemented correctly. If not the team must review the source of the failure.
    - Actual Results
        - All algorithm scripts acted as expected and were able to detect their respectives anomalies.

Complete platform

- Event Simulations
    - Description
        - The team tested the events that were described in the individual algorithms, but utilizing the entire system.
    - Procedures
        - The team wrote a unit test to simulate the anomalies proposed and see if the system reacts accordingly.
    - Expected Results
        - The test is expected to be successful all the previous modules were implemented correctly. If not the team must review the source of the failure.
    - Actual Results
        - The entire system worked as expected.

Appendix H: Economic Analysis

Human Resources

The total Expenditure of human resources were $63,062.16 dollars. The project concluded with a total of 13 weeks of work as anticipated. The projected expenditure was of $41,496.90 dollars. The team concluded over budget by $21,565.26 dollars. This excess is due to misunderstandings with the dates and required material for the dates. The misunderstanding caused the team to move 1 week of work ahead of time to meet one of the deadlines. Because of this, the team had to put in extra hours to meet the deadline of the proposal delivery. The team also end up working more hours than anticipated during the project to ensure the proper delivery of the system. The team also benefited from the fact that the consultants were not needed as expected. Also, the full payment of the fringe benefits, a total of $3,535.95, were included in both expenditures.

Name	Role	Stipend	Estimated Working Hours per Week	Total Weeks	Total Projected Wages	Current Worked Hours	Overtime Hours (1.5x)	Total Expenditure
Marie A. Nazario	Software Engineer / Project Manager	$ 24.93/hr	30 hr/week	13	$ 9,722.70	728	42	$ 19,719.63
Pedro Colón	Software Engineer	$ 24.16/hr	30 hr/week	13	$ 9,422.40	672	15	$ 16,779.12
Antoine Cotto	Software Engineer	$ 24.16/hr	30 hr/week	13	$ 9,422.40	768	56	$ 20,584.32
Luis Lugo	System Admin. / Consultant	$ 36.45/hr	3 hr/week	11	$ 1,202.85	18	0	$ 656.10
Emmanuel Arzuaga	Consultant	$ 49.64/hr	3 hr/week	11	$ 1,638.12	1	0	$ 49.64
Nayda Santiago	Consultant	$ 49.64/hr	3 hr/week	11	$ 1,638.12	16	0	$ 794.24
Jose F. Vega	Consultant	$ 49.64/hr	3 hr/week	11	$ 1,638.12	1	0	$ 49.64
Isidoro Couvertier	Consultant	$ 49.64/hr	3 hr/week	11	$ 1,638.12	6	0	$ 297.84
Suresh Damodaran	Consultant	$ 49.64/hr	3 hr/week	11	$ 1,638.12	12	0	$ 595.68
				Total Cost :	$ 37,960.95		**Total Cost :**	$ 59,526.21

Table 14: Labor Costs

Equipment

The team incurred in the cost of buying work laptop computers, server time and network switches. Incurring in the total projected expenditure.

Equipment	Price	Quantity	Projected Expenditure	Total Expenditure
Server	$ 24.56 per month per node	15 nodes, 2 months	$ 736.80	$ 736.80
Network Switch	$ 789.75	2	$ 1,579.50	$ 1,579.50
Laptop Computer	$ 1,499.00	3	$ 4,500.00	$ 4,500.00
		Total:	$ 6,816.30	$ 6,816.30

Table 15: Labor Costs

Expenditure Analysis

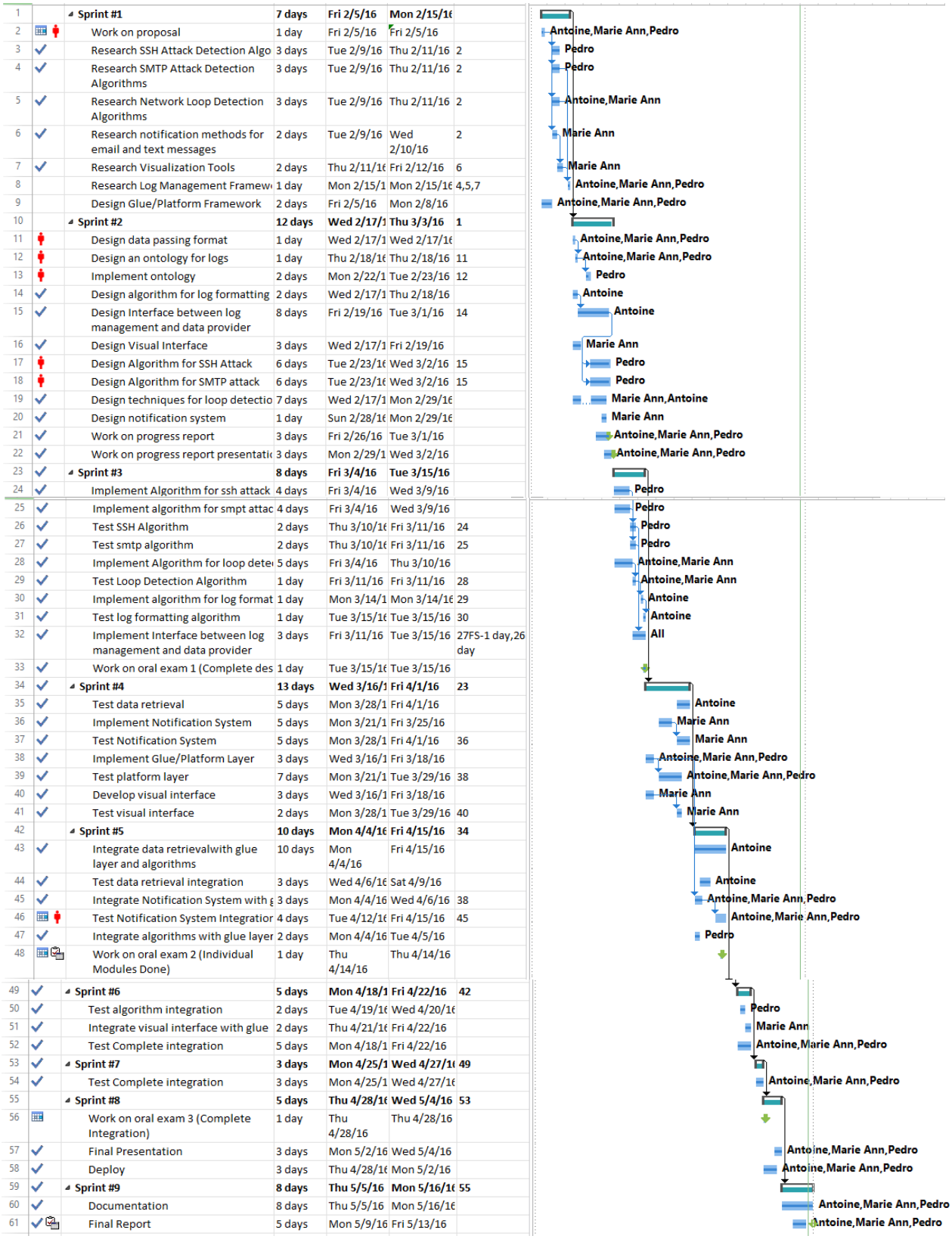
The team concluded with a final expenditure of $69,878.46 dollars. Although this is higher than the projected budget ($41,496.90 dollars) by $21,565.26, technically being over budget, we are still covered by our overhead cost and such under the budget allocated to the project.

Appendix I: Task Progress and Gantt Chart

The link to the NADS project Gantt Chart is the following:

https://drive.google.com/a/upr.edu/file/d/0B1MB9uu1QJwyZTI5Z084bHFubDg/view?usp=sharing

We also included images of the gantt chart for a quick overview of it.



1		Sprint #1	7 days	Fri 2/5/16	Mon 2/15/16	
2		Work on proposal	1 day	Fri 2/5/16	Fri 2/5/16	
3		Research SSH Attack Detection Algo	3 days	Tue 2/9/16	Thu 2/11/16	2
4		Research SMTP Attack Detection Algorithms	3 days	Tue 2/9/16	Thu 2/11/16	2
5		Research Network Loop Detection Algorithms	3 days	Tue 2/9/16	Thu 2/11/16	2
6		Research notification methods for email and text messages	2 days	Tue 2/9/16	Wed 2/10/16	2
7		Research Visualization Tools	2 days	Thu 2/11/16	Fri 2/12/16	6
8		Research Log Management Framew	1 day	Mon 2/15/1	Mon 2/15/16	4,5,7
9		Design Glue/Platform Framework	2 days	Fri 2/5/16	Mon 2/8/16	
10		Sprint #2	12 days	Wed 2/17/1	Thu 3/3/16	1
11		Design data passing format	1 day	Wed 2/17/1	Wed 2/17/16	
12		Design an ontology for logs	1 day	Thu 2/18/16	Thu 2/18/16	11
13		Implement ontology	2 days	Mon 2/22/1	Tue 2/23/16	12
14		Design algorithm for log formatting	2 days	Wed 2/17/1	Thu 2/18/16	
15		Design Interface between log management and data provider	8 days	Fri 2/19/16	Tue 3/1/16	14
16		Design Visual Interface	3 days	Wed 2/17/1	Fri 2/19/16	
17		Design Algorithm for SSH Attack	6 days	Tue 2/23/16	Wed 3/2/16	15
18		Design Algorithm for SMTP attack	6 days	Tue 2/23/16	Wed 3/2/16	15
19		Design techniques for loop detectio	7 days	Wed 2/17/1	Mon 2/29/16	
20		Design notification system	1 day	Sun 2/28/16	Mon 2/29/16	
21		Work on progress report	3 days	Fri 2/26/16	Tue 3/1/16	
22		Work on progress report presentatic	3 days	Mon 2/29/1	Wed 3/2/16	
23		Sprint #3	8 days	Fri 3/4/16	Tue 3/15/16	
24		Implement Algorithm for ssh attack	4 days	Fri 3/4/16	Wed 3/9/16	
25		Implement algorithm for smpt attac	4 days	Fri 3/4/16	Wed 3/9/16	
26		Test SSH Algorithm	2 days	Thu 3/10/16	Fri 3/11/16	24
27		Test smtp algorithm	2 days	Thu 3/10/16	Fri 3/11/16	25
28		Implement Algorithm for loop dete	5 days	Fri 3/4/16	Thu 3/10/16	
29		Test Loop Detection Algorithm	1 day	Fri 3/11/16	Fri 3/11/16	28
30		Implement algorithm for log format	1 day	Mon 3/14/1	Mon 3/14/16	29
31		Test log formatting algorithm	1 day	Tue 3/15/16	Tue 3/15/16	30
32		Implement Interface between log management and data provider	3 days	Fri 3/11/16	Tue 3/15/16	27FS-1 day,26 day
33		Work on oral exam 1 (Complete des	1 day	Tue 3/15/16	Tue 3/15/16	
34		Sprint #4	13 days	Wed 3/16/1	Fri 4/1/16	23
35		Test data retrieval	5 days	Mon 3/28/1	Fri 4/1/16	
36		Implement Notification System	5 days	Mon 3/21/1	Fri 3/25/16	
37		Test Notification System	5 days	Mon 3/28/1	Fri 4/1/16	36
38		Implement Glue/Platform Layer	3 days	Wed 3/16/1	Fri 3/18/16	
39		Test platform layer	7 days	Mon 3/21/1	Tue 3/29/16	38
40		Develop visual interface	3 days	Wed 3/16/1	Fri 3/18/16	
41		Test visual interface	2 days	Mon 3/28/1	Tue 3/29/16	40
42		Sprint #5	10 days	Mon 4/4/16	Fri 4/15/16	34
43		Integrate data retrievalwith glue layer and algorithms	10 days	Mon 4/4/16	Fri 4/15/16	
44		Test data retrieval integration	3 days	Wed 4/6/16	Sat 4/9/16	
45		Integrate Notification System with g	3 days	Mon 4/4/16	Wed 4/6/16	38
46		Test Notification System Integratior	4 days	Tue 4/12/16	Fri 4/15/16	45
47		Integrate algorithms with glue layer	2 days	Mon 4/4/16	Tue 4/5/16	
48		Work on oral exam 2 (Individual Modules Done)	1 day	Thu 4/14/16	Thu 4/14/16	
49		Sprint #6	5 days	Mon 4/18/1	Fri 4/22/16	42
50		Test algorithm integration	2 days	Tue 4/19/16	Wed 4/20/16	
51		Integrate visual interface with glue	2 days	Thu 4/21/16	Fri 4/22/16	
52		Test Complete integration	5 days	Mon 4/18/1	Fri 4/22/16	
53		Sprint #7	3 days	Mon 4/25/1	Wed 4/27/16	49
54		Test Complete integration	3 days	Mon 4/25/1	Wed 4/27/16	
55		Sprint #8	5 days	Thu 4/28/16	Wed 5/4/16	53
56		Work on oral exam 3 (Complete Integration)	1 day	Thu 4/28/16	Thu 4/28/16	
57		Final Presentation	3 days	Mon 5/2/16	Wed 5/4/16	
58		Deploy	3 days	Thu 4/28/16	Mon 5/2/16	
59		Sprint #9	8 days	Thu 5/5/16	Mon 5/16/16	55
60		Documentation	8 days	Thu 5/5/16	Mon 5/16/16	
61		Final Report	5 days	Mon 5/9/16	Fri 5/13/16	

Figure 20: Gantt Chart

Appendix J: Ethical Analysis of Project

In this project there is one main ethical problem which is that of the safety of the users in the system. The network of the ECE department at any moment is vulnerable to SSH and SMTP protocols attack that aim to subvert its security. Through the use of NADS we aim to reduce the likelihood of these attacks being successful. Although this is the aim of the system, we have the problem that the system may fail or that we may have false positives and block real users of the network. In this conflict the main affected parties are: the users of the system, the system administrator and the possible system attackers. To resolve the issue of having false positives on the system our solutions are to either ignore the problem in favor of better security or to configure the parameters in such a way that a happy medium can be reached between very high security and very lax security. We found believe that happy medium is the best solution as it provides an acceptable level of harm to the user in the form of a less secure algorithms. When using the point of view of user for reversibility we see this as the best option too, as we don't want to be blocked because of our everyday casual use. In the case of the conflict that the system may fail we have no other choice but to try and make the system as foolproof as possible.

Appendix K: KQL-ELK Layer

Ontology

The team decided to use a data abstraction mechanism that relied on the implementation of an ontology. This ontology maps the physical fields in the data with the concepts/types and interpretations behind them.

Sample Log Data

The team was given the following sample log data.

Feb 2 16:35:32 ece SSHd[8213]: Failed none for invalid user estudiante from 136.145.116.202 port 61808 SSH2

Feb 2 17:09:08 ece SSHd[9958]: refused connect from 61.175.198.188 (61.175.198.188)

Feb 2 17:31:30 ece SSHd[3521]: pam_unix(SSHd:session): session closed for user domingo

Feb 2 17:34:41 ece SSHd[11335]: reverse mapping checking getaddrinfo for catv.choicecable.net [206.248.70.141] failed - POSSIBLE BREAK-IN ATTEMPT!

Feb 2 17:34:41 ece SSHd[11335]: Accepted password for cruzpol from 206.248.70.141 port 61737 SSH2

Extracted Fields

The team was able to extract the following fields from the aforementioned data:

- Date
- HostName
- DaemonName
- DaemonPort
- Status
- ClientIp
- UserName
- ClientPort
- ClientDaemon

Dimensions, Tags, and Dimension Sets

The team was then able to extract some dimensions (types) from this data and join them in a dimension set called login that represents a message from a login. The extracted dimensions, which are representations of the types of the data, are datetime, name, protocol, port number, status, id, ip_address, and domain_name. These dimensions abstract the physical fields of the data. Additionally the team was able to generate some tags to give a specific meaning to dimensions that may contain data of different interpretations. These tags were: servername, client, host, and user. For a complete explanation of the Dimensions, Tags and Dimension Sets view the technical report on Address expressions and KQL. The team consulted with Dr. Damodaran to verify that the Dimensions, Tags, and DimensionSets were developed following the best principles possible.

For more information refer to [29].

Ontology Generation Process

The following is the process that the team followed to be able to develop the KQL ontology. It is worth mentioning that some of the steps in the process may not be necessary if the data is previously structured.

1. The team took a sample of the log output that needed to be searchable through KQL.
2. The team then mapped the data in the log messages to a group of fields.
    a. Date
    b. HostName
    c. DaemonName
    d. DaemonPort
    e. Status
    f. ClientIp
    g. UserName
    h. ClientPort
    i. ClientDaemon
3. The team proceeded to analyze the type of the data within the fields.
    a. The team tried to find the general types of the data. An example of this generation of general types was the field Date which contained a calendar date. The type of the data within this field was of the datetime type as they were dates of a specific time.
4. The team was able to extract an initial set of Dimensions (types) which mapped the fields to the data.

a. datetime
b. name
c. protocol
d. portnumber
e. status
f. id
g. ip_address
h. domain_name

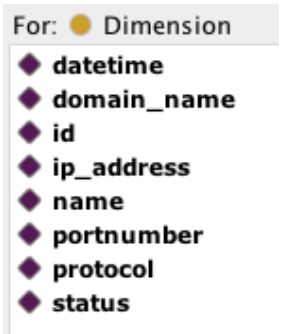
5. The team then proceeded to evaluate the contexts and see if one of the types had more than one possible use.
   a. An example is the Dimension ip_address which could be used in the context of a host and a client.
6. The team then proceeded to generate a list of the Tags (contexts) that the Dimensions could have:
   a. servername
   b. client
   c. host
   d. user
   e. client
7. The team then proceeded to group the Dimensions into a Dimension Set that would represent a typical login message.

Final Ontology

The final ontology that the team utilized was done in Protégé.

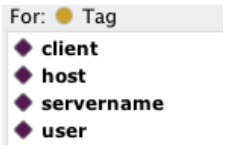
Dimensions

The following dimensions were developed to encompass the data fields. To view which dimension contains which field, view the Usage section.



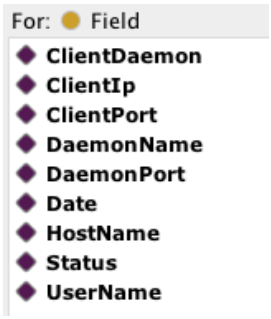
Tags

The following tags were developed to target specific usages of data inside the dimension. To view which fields and dimensions the tags were targeted for, view the Usage section.



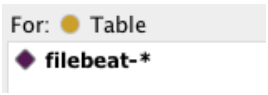
Fields

The following fields are the ones that were derived from the data.



Tables

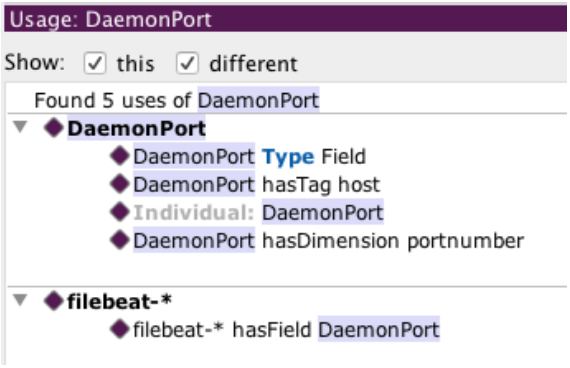
The data fields resided within the following tables.



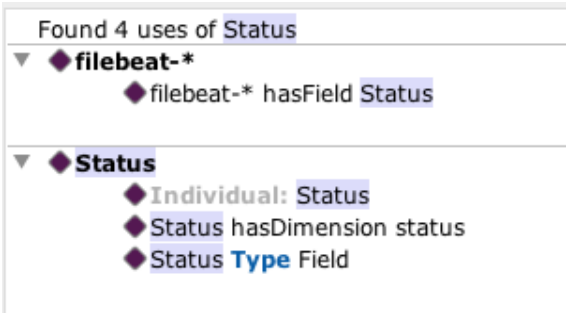
Usages

The following figures show the fields and which dimensions, tags, and tables they belonged/were mapped to.

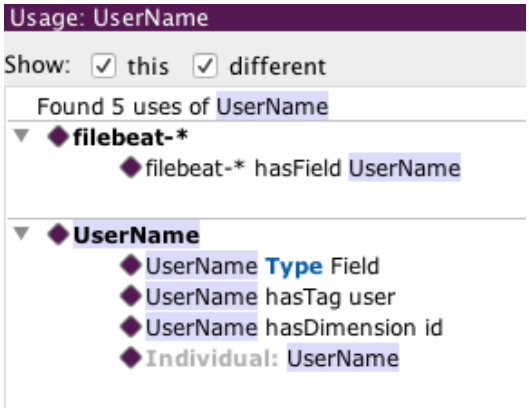
- DaemonPort



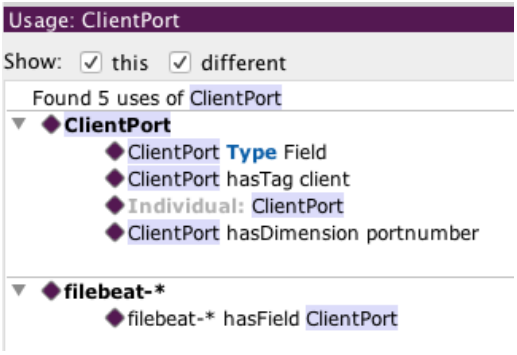
- Status



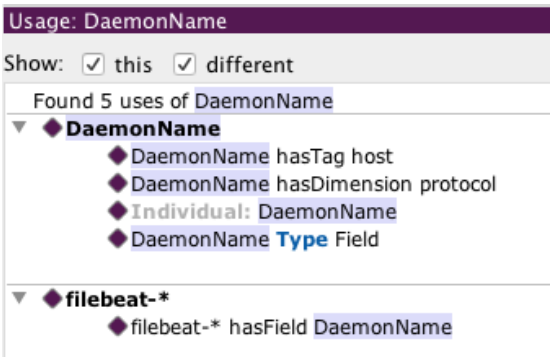
- UserName



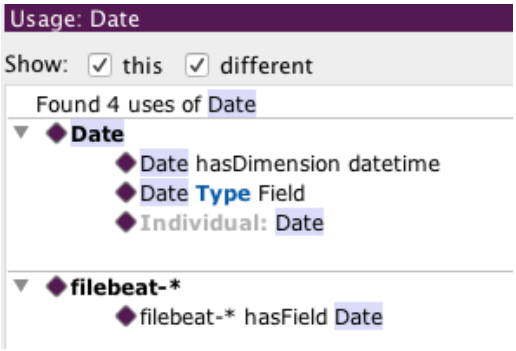
- ClientPort



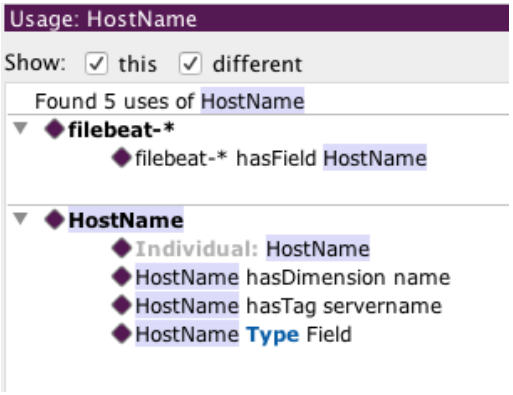
- DaemonName



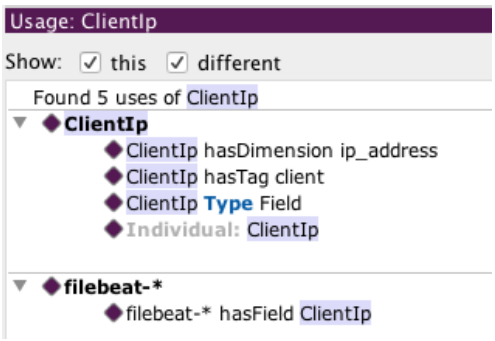
- Date



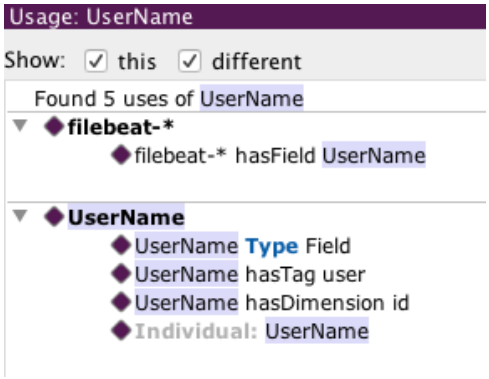
- HostName



- ClientIp



- UserName



Sample Queries

Some of the queries used in the SSH/SMTP Protocol Attack Detection Algorithm are the following:

1. Get Successful Logins for specified protocol
    a. 'SELECT \ ALL*{protocol,portnumber,status,id,ip_address,datetime} \ from \ ALL/{protocol,portnumber,status,id,ip_address} \ where (\ ALL*status \ like '*Accepted*') and \ ALL*protocol*_:host \ like '*sshd*'"
2. Get all login attempts for specified protocol
    a. 'SELECT \ ALL*{protocol,portnumber,status,id,ip_address,datetime,name} \ from \ ALL/{protocol,portnumber,status,id,ip_address} \ where \ ALL*name*_:servername \ like "ubuntu" and \ ALL*protocol*_:host \ like '*sshd*' and (\ ALL*status \ like '*ailed*' or \ ALL*status \ like '*Accepted*') '

Performance

A performance evaluation was run to assess the impact of the integration of KQL into the Elasticsearch-SQL plugin. To test the performance, a KQL converted query was introduced into the Elasticsearch-SQL plugin. The time it took to execute a KQL query and a SQL query were utilized for the evaluation.

Elasticsearch-SQL Performance

Output for Example Query 1 Translated Directly to SQL:

DEBUG:root:Time it took to make call:0:00:01.413222For query:SELECT HostName,UserName,ClientDaemon,Status,ClientPort,Date,DaemonName,ClientIp,

DaemonPort from filebeat-* where HostName like "ubuntu" and DaemonName like "*sshd*" and (Status like "*ailed*" or Status like "*Accepted*")
DEBUG:root:Time it took to make call:0:00:00.054700For query:SELECT HostName,UserName,ClientDaemon,Status,ClientPort,Date,DaemonName,ClientIp, DaemonPort from filebeat-* where HostName like "ubuntu" and DaemonName like "*sshd*" and (Status like "*ailed*" or Status like "*Accepted*")
DEBUG:root:Time it took to make call:0:00:00.045174For query:SELECT HostName,UserName,ClientDaemon,Status,ClientPort,Date,DaemonName,ClientIp, DaemonPort from filebeat-* where HostName like "ubuntu" and DaemonName like "*sshd*" and (Status like "*ailed*" or Status like "*Accepted*")
DEBUG:root:Time it took to make call:0:00:00.038170For query:SELECT HostName,UserName,ClientDaemon,Status,ClientPort,Date,DaemonName,ClientIp, DaemonPort from filebeat-* where HostName like "ubuntu" and DaemonName like "*sshd*" and (Status like "*ailed*" or Status like "*Accepted*")
DEBUG:root:Time it took to make call:0:00:00.033964For query:SELECT HostName,UserName,ClientDaemon,Status,ClientPort,Date,DaemonName,ClientIp, DaemonPort from filebeat-* where HostName like "ubuntu" and DaemonName like "*sshd*" and (Status like "*ailed*" or Status like "*Accepted*")
DEBUG:root:Time it took to make call:0:00:00.044023For query:SELECT HostName,UserName,ClientDaemon,Status,ClientPort,Date,DaemonName,ClientIp, DaemonPort from filebeat-* where HostName like "ubuntu" and DaemonName like "*sshd*" and (Status like "*ailed*" or Status like "*Accepted*")
DEBUG:root:Time it took to make call:0:00:00.029303For query:SELECT HostName,UserName,ClientDaemon,Status,ClientPort,Date,DaemonName,ClientIp, DaemonPort from filebeat-* where HostName like "ubuntu" and DaemonName like "*sshd*" and (Status like "*ailed*" or Status like "*Accepted*")
DEBUG:root:Time it took to make call:0:00:00.021913For query:SELECT HostName,UserName,ClientDaemon,Status,ClientPort,Date,DaemonName,ClientIp, DaemonPort from filebeat-* where HostName like "ubuntu" and DaemonName like "*sshd*" and (Status like "*ailed*" or Status like "*Accepted*")

From the output we extract the following:

Query	Time
SELECT	1.413222

HostName,UserName,ClientDaemon,Status,ClientPort,Date,DaemonName,ClientIp,DaemonPort from filebeat-* where HostName like "ubuntu" and DaemonName like "*sshd*" and (Status like "*ailed*" or Status like "*Accepted*")	0.054700
	0.045174
	0.038170
	0.033964
	0.044023
	0.029303
	0.021913

Output for Example Query 2 Translated Directly to SQL:

DEBUG:root:Time it took to make call:0:00:00.019642For query:SELECT UserName,ClientDaemon,Status,ClientPort,Date,DaemonName,ClientIp,DaemonPort from filebeat-* where (Status like "*Accepted*") and DaemonName like "*sshd*"
DEBUG:root:Time it took to make call:0:00:00.025262For query:SELECT UserName,ClientDaemon,Status,ClientPort,Date,DaemonName,ClientIp,DaemonPort from filebeat-* where (Status like "*Accepted*") and DaemonName like "*sshd*"
DEBUG:root:Time it took to make call:0:00:00.017454For query:SELECT UserName,ClientDaemon,Status,ClientPort,Date,DaemonName,ClientIp,DaemonPort from filebeat-* where (Status like "*Accepted*") and DaemonName like "*sshd*"
DEBUG:root:Time it took to make call:0:00:00.011982For query:SELECT UserName,ClientDaemon,Status,ClientPort,Date,DaemonName,ClientIp,DaemonPort from filebeat-* where (Status like "*Accepted*") and DaemonName like "*sshd*"
DEBUG:root:Time it took to make call:0:00:00.024122For query:SELECT UserName,ClientDaemon,Status,ClientPort,Date,DaemonName,ClientIp,DaemonPort from filebeat-* where (Status like "*Accepted*") and DaemonName like "*sshd*"
DEBUG:root:Time it took to make call:0:00:00.029700For query:SELECT UserName,ClientDaemon,Status,ClientPort,Date,DaemonName,ClientIp,DaemonPort from filebeat-* where (Status like "*Accepted*") and DaemonName like "*sshd*"
DEBUG:root:Time it took to make call:0:00:00.047182For query:SELECT UserName,ClientDaemon,Status,ClientPort,Date,DaemonName,ClientIp,DaemonPort from filebeat-* where (Status like "*Accepted*") and DaemonName like "*sshd*"

DEBUG:root:Time it took to make call:0:00:00.069175For query:SELECT UserName,ClientDaemon,Status,ClientPort,Date,DaemonName,ClientIp,DaemonPort from filebeat-* where (Status like "*Accepted*") and DaemonName like "*sshd*"

From the output we extract the following:

Query	Time
SELECT UserName,ClientDaemon,Status,ClientPort,Date,DaemonName,ClientIp,DaemonPort from filebeat-* where (Status like "*Accepted*") and DaemonName like "*sshd*"	0.019642
	0.025262
	0.017454
	0.011982
	0.024122
	0.029700
	0.047182
	0.069175

Elasticsearch-KQL Performance

Output for Example Query 1 Translated to SQL by KQL:

DEBUG:root:Time it took to make call:0:00:02.928018For query:SELECT \ ALL*{protocol,portnumber,status,id,ip_address,datetime,name} \ from \ ALL/{protocol,portnumber,status,id,ip_address} \ where \ ALL*name*_:servername \ like "ubuntu" and \ ALL*protocol*_:host \ like "*sshd*" and (\ ALL*status \ like "*ailed*" or \ ALL*status \ like "*Accepted*")
DEBUG:root:Time it took to make call:0:00:00.149739For query:SELECT \ ALL*{protocol,portnumber,status,id,ip_address,datetime,name} \ from \ ALL/{protocol,portnumber,status,id,ip_address} \ where \ ALL*name*_:servername \ like "ubuntu" and \ ALL*protocol*_:host \ like "*sshd*" and (\ ALL*status \ like "*ailed*" or \ ALL*status \ like "*Accepted*")
DEBUG:root:Time it took to make call:0:00:00.230106For query:SELECT \ ALL*{protocol,portnumber,status,id,ip_address,datetime,name} \ from \

ALL/{protocol,portnumber,status,id,ip_address} \ where \ ALL*name*_:servername \ like "ubuntu" and \ ALL*protocol*_:host \ like "*sshd*" and (\ ALL*status \ like "*ailed*" or \ ALL*status \ like "*Accepted*")
DEBUG:root:Time it took to make call:0:00:00.233418For query:SELECT \ ALL*{protocol,portnumber,status,id,ip_address,datetime,name} \ from \ ALL/{protocol,portnumber,status,id,ip_address} \ where \ ALL*name*_:servername \ like "ubuntu" and \ ALL*protocol*_:host \ like "*sshd*" and (\ ALL*status \ like "*ailed*" or \ ALL*status \ like "*Accepted*")
DEBUG:root:Time it took to make call:0:00:00.369946For query:SELECT \ ALL*{protocol,portnumber,status,id,ip_address,datetime,name} \ from \ ALL/{protocol,portnumber,status,id,ip_address} \ where \ ALL*name*_:servername \ like "ubuntu" and \ ALL*protocol*_:host \ like "*sshd*" and (\ ALL*status \ like "*ailed*" or \ ALL*status \ like "*Accepted*")
DEBUG:root:Time it took to make call:0:00:00.209721For query:SELECT \ ALL*{protocol,portnumber,status,id,ip_address,datetime,name} \ from \ ALL/{protocol,portnumber,status,id,ip_address} \ where \ ALL*name*_:servername \ like "ubuntu" and \ ALL*protocol*_:host \ like "*sshd*" and (\ ALL*status \ like "*ailed*" or \ ALL*status \ like "*Accepted*")
DEBUG:root:Time it took to make call:0:00:00.401961For query:SELECT \ ALL*{protocol,portnumber,status,id,ip_address,datetime,name} \ from \ ALL/{protocol,portnumber,status,id,ip_address} \ where \ ALL*name*_:servername \ like "ubuntu" and \ ALL*protocol*_:host \ like "*sshd*" and (\ ALL*status \ like "*ailed*" or \ ALL*status \ like "*Accepted*")
DEBUG:root:Time it took to make call:0:00:00.402045For query:SELECT \ ALL*{protocol,portnumber,status,id,ip_address,datetime,name} \ from \ ALL/{protocol,portnumber,status,id,ip_address} \ where \ ALL*name*_:servername \ like "ubuntu" and \ ALL*protocol*_:host \ like "*sshd*" and (\ ALL*status \ like "*ailed*" or \ ALL*status \ like "*Accepted*")

From the output we extract the following:

Query	Time
SELECT \ ALL*{protocol,portnumber,status,id,ip_address,datetime,name} \ from \ ALL/{protocol,portnumber,status,id,ip_ad	2.928018
	0.149739
	0.230106

dress} \ where \ ALL*name*_:servername \ like "ubuntu" and \ ALL*protocol*_:host \ like "*sshd*" and (\ ALL*status \ like "*ailed*" or \ ALL*status \ like "*Accepted*")	0.233418
	0.369946
	0.209721
	0.401961
	0.402045

Output for Example Query 2 Translated to SQL by KQL:

DEBUG:root:Time it took to make call:0:00:00.131866For query:SELECT \ ALL*{protocol,portnumber,status,id,ip_address,datetime} \ from \ ALL/{protocol,portnumber,status,id,ip_address} \ where (\ ALL*status \ like "*Accepted*") and \ ALL*protocol*_:host \ like "*sshd*"
DEBUG:root:Time it took to make call:0:00:00.133549For query:SELECT \ ALL*{protocol,portnumber,status,id,ip_address,datetime} \ from \ ALL/{protocol,portnumber,status,id,ip_address} \ where (\ ALL*status \ like "*Accepted*") and \ ALL*protocol*_:host \ like "*sshd*"
DEBUG:root:Time it took to make call:0:00:00.110085For query:SELECT \ ALL*{protocol,portnumber,status,id,ip_address,datetime} \ from \ ALL/{protocol,portnumber,status,id,ip_address} \ where (\ ALL*status \ like "*Accepted*") and \ ALL*protocol*_:host \ like "*sshd*"
DEBUG:root:Time it took to make call:0:00:00.082998For query:SELECT \ ALL*{protocol,portnumber,status,id,ip_address,datetime} \ from \ ALL/{protocol,portnumber,status,id,ip_address} \ where (\ ALL*status \ like "*Accepted*") and \ ALL*protocol*_:host \ like "*sshd*"
DEBUG:root:Time it took to make call:0:00:00.128448For query:SELECT \ ALL*{protocol,portnumber,status,id,ip_address,datetime} \ from \ ALL/{protocol,portnumber,status,id,ip_address} \ where (\ ALL*status \ like "*Accepted*") and \ ALL*protocol*_:host \ like "*sshd*"
DEBUG:root:Time it took to make call:0:00:00.045754For query:SELECT \ ALL*{protocol,portnumber,status,id,ip_address,datetime} \ from \ ALL/{protocol,portnumber,status,id,ip_address} \ where (\ ALL*status \ like "*Accepted*") and \ ALL*protocol*_:host \ like "*sshd*"
DEBUG:root:Time it took to make call:0:00:00.040770For query:SELECT \ ALL*{protocol,portnumber,status,id,ip_address,datetime} \ from \

ALL/{protocol,portnumber,status,id,ip_address} \ where (\ ALL*status \ like "*Accepted*") and \ ALL*protocol*_:host \ like "*sshd*"
DEBUG:root:Time it took to make call:0:00:00.137274For query:SELECT \ ALL*{protocol,portnumber,status,id,ip_address,datetime} \ from \ ALL/{protocol,portnumber,status,id,ip_address} \ where (\ ALL*status \ like "*Accepted*") and \ ALL*protocol*_:host \ like "*sshd*"

From the output we extract the following:

Query	Time
SELECT \ ALL*{protocol,portnumber,status,id,ip_address,datetime} \ from \ ALL/{protocol,portnumber,status,id,ip_address} \ where (\ ALL*status \ like "*Accepted*") and \ ALL*protocol*_:host \ like "*sshd*"	0.131866
	0.133549
	0.110085
	0.082998
	0.128448
	0.045754
	0.040770
	0.137274

Comparison

Output 1:

SQL	KQL	% Diff
1.413222	2.928018	69.78632833
0.0547	0.149739	92.97541076
0.045174	0.230106	134.3591979
0.03817	0.233418	143.7824941

0.033964	0.369946	166.3647842
0.044023	0.209721	130.602497
0.029303	0.401961	172.8212881
0.021913	0.402045	179.3253105

Average % Diff: 136.2521639

Output 2:

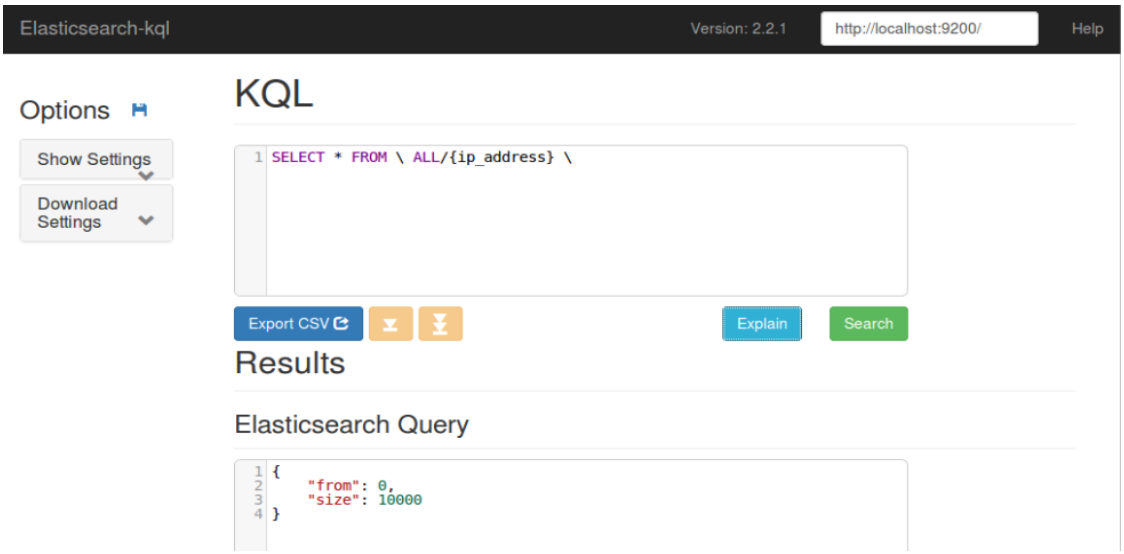
SQL	KQL	% Diff
0.019642	0.131866	148.1426723
0.025262	0.133549	136.3721657
0.017454	0.110085	145.2590972
0.011982	0.082998	149.5388503
0.024122	0.128448	136.7582093
0.0297	0.045754	42.5530787
0.047182	0.04077	14.58068037
0.069175	0.137274	65.9717412

Average % Diff: 104.8970619

The usage of KQL creates approximately a 120% difference. This is a large performance difference, however we are talking about timings that are well below a second. This difference could be attributed to the usage of a remote PHP-SQL parser. The Elasticsearch plugin that was extended could not execute a call to PHP because of security permissions. To work around this, a PHP dependency that the KQL processor has had to be moved to a remote Nginx server and accessed through a rest request. This additional request may have a larger overhead than the plain request that the original elasticsearch-sql has. The performance difference could also be attributed to the fact that the Elasticsearch-SQL plugin does not have to access files or the security manager to perform its duties, something that the new Elasticsearch-KQL plugin has to do.

KQL Visual Interface

Below is an example of the prototype interface that is being developed to simplify accessing/configuring and querying data through KQL.



There is a query area, an explain button which shows the resolution of the KQL query and the details of the Elasticsearch query that was executed. This is an experimental interface which is being developed to have the ability to edit the configuration/mapping files of the language, autocompletion help for the language, and better error reporting in the case of a failed query.

Development Notes

See KQL's git readme for further details