TOWSON UNIVERSITY

OFFICE OF GRADUATE STUDIES

A HYBRID INTELLIGENCE/MULTI-AGENT SYSTEM FOR MINING INFORMATION ASSURANCE DATA

By

Charles A. Fowler

A Dissertation

Presented to the faculty of Towson University

in partial fulfillment of the requirements for the degree

Doctor of Science

Department of Computer and Information Sciences

Towson University

Towson, Maryland 21252

May 2015

TOWSON UNIVERSITY
OFFICE OF GRADUATE STUDIES

DISSERTATION APPROVAL PAGE

This is to certify that the dissertation prepared by **Charles A. Fowler** entitled

A HYBRID INTELLIGENCE/MULTI-AGENT SYSTEM FOR MINING INFORMATION ASSURANCE DATA has been approved by the thesis committee as satisfactorily completing the dissertation requirements for the degree **Doctor of Science**.

Chairperson, Dissertation Committee: Robert J. Hammell II — Date: 4/21/15

Committee Member: Subrata Acharya — Date: 4/21/2015

Committee Member: Sidd Kaza — Date: 4/21/2015

Committee Member: Wei Yu — Date: 4/21/15

Dean of Graduate Studies — Date: 4-27-15

Acknowledgements

I would like to thank my committee members: Dr. Subrata Acharya, Dr. Sidd Kaza, and Dr. Wei Yu who offered timely insight, guidance, and direction which was essential for the completion of this endeavor.

Words escape me in my efforts to express the extent of my gratitude to Dr. Hammell who served as my dissertation committee chair and trusted advisor throughout this entire undertaking. Without his diligent oversight and even handed shepherding I'm sure I would have made it far, but I would not have ended up where I belonged.

I am exceedingly grateful to my wonderful parents, devoted sons, and my loving and always supportive wife. I ask their forgiveness again for the many times they graciously acquiesced as I answered their queries with "So sorry, not just now, maybe in a little bit…" and put my nose back to this effort. They always offered their support and encouragement through the not insignificant time this task took.

Abstract

A hybrid intelligence/multi-agent system for mining information assurance data

Charles A. Fowler

Organizations across all domains and of all sizes wrestle with the problem of "coping with information overload," or CwIO. They ingest more and more data, in new and varied formats every day, and struggle increasingly vigorously to find the nuggets of knowledge hidden within the vast amounts of information. Furthermore, due to the various and pervasive types of noise in the haystack of data, it is becoming increasingly difficult to discern between shiny false shards and the true needles of knowledge. Although the costs of data storage, memory and processing have dropped, this decline has not maintained parity with the unprecedented increase in the amount and complexity of data to be examined. This problem is especially challenging in the world of network intrusion detection. In this realm, one must not only deal with sifting through vast amounts of data, but it must also be done in a timely manner even when at times one is not sure what exactly it is being sought. In efforts to help solve this problem, this research demonstrates that in the realm of offline network forensic datamining, several different datamining algorithms (hybrid intelligence) working within a multi-agent system, will yield more accurate results than any one datamining algorithm acting on its own. Toward that end, this paper outlines the steps taken to generate and prepare suitably minable test data, compare and contrast the capabilities/output of various types of datamining algorithms (hybrid intelligence), and finally discuss the architecture and construction of a SPADE based multi-agent system to semi-autonomously perform multi-path datamining tasks.

Table of Contents

List of Figures

Overview of the chapters

This dissertation is organized as follows: In the Introduction and Background, we outline and discuss the issue we set out to help solve, the "Coping with Information Overload" problem. Then we briefly touch upon the progression of historical answers that others have devised to tackle it, the foundations upon which we have built our solution. We discuss the benefits of a cross-domain, interdisciplinary approach to solving the issue, and introduce our solution built on just such a framework. We discuss our chosen domain of intrusion detection and touch upon some of the unexpected problems we encountered therein. We wrap up this section by enumerating the expectations we had at the outset of this research.

In the Literature Review we delve deeper into our examinations of the various concepts introduced in the first section, specifically: datamining, intrusion detection, and multi-agent systems. We also look at both the benefits and barriers to cross pollinizing solution spaces, while finishing out with an enumeration of tools necessary to duplicate our research.

In the Approach we take an in depth walk through the steps undertaken to complete this research. We start by showing how to initially generate data which recreates the problems discussed in the previous section, and then craft a novel hybrid intelligence solution, woven together in a multi-agent system. As reiterated below, one of the chief aims of this undertaking was not only to provide results demonstrating the benefits of a new methodology, but to also develop and document repeatable procedures which would make these benefits relevant and tractable to the everyday network and security administrator.

In the Results section we provide a detailed look at the outcomes of our experiments and examine data which supports the necessity of having taken a side trip along the way.

By way of the Conclusion section, we recapitulate the research questions and enumerate the answers to them which were discovered through our research. We also reexamine the measures of success and report on how the results of this work bears on our thesis.

We wrap up the discussions with an eye towards the future, looking at potential follow on paths for this research. We then discuss the portability of our solution, not only within our research domain (intrusion detection), but also its applicability in other problem spaces.

I. INTRODUCTION AND BACKGROUND

Covered in this chapter…

- What the problem is and why it matters
- Historical solutions
- Cross domain concepts

A. Defining the problem

1) Coping with Information Overload (CwIO)

The "needles" of knowledge are in there, all the while, more and more "hay" piles up in the fields of data, more than anyone can possibly sift through in a timely manner, and any knowledge contained in that data grows increasingly stale as the minutes whiz by. On our battlefields, national borders, college campuses, and city streets we have deployed a greater number of cameras, microphones, antennas, multi-sensor equipped drones, information capture and logging devices than ever before. Our ability to gather and ingest more types and greater amounts of data increases, while our ability to cull useful nuggets of actionable information from therein decreases. [1] Not only is the volume of data problematic, but the knowledge buried within it is woven in intricate patterns at multiple levels, and are not always evident from a head on, top or side view. Segaran elucidates how the problem manifests across a variety of disciplines:

> …knowledge of machine-learning algorithms can be helpful for software developers in many other fields. They are particularly useful in areas that deal with large datasets that can be searched for interesting patterns. [2]

He goes on to mention a handful of fields which benefit from datamining, including: biotechnology, financial fraud detection, machine vision, product marketing, supply chain optimization, stock market analysis and national security, just to name a few. [2]

While there is no question that the coping with information overload problem exists, there is also no question that current solutions are unsatisfactory, and researchers are constantly invoking unique and new ways to attack the problem [3] [4] [5] [6] [7]. Companies like Netflix and Microsoft for example even sponsor datamining contests in efforts to drum up new and unique solutions to this problem [8] [9]. The constant increase in volume, complexity and lack of homogeneity of data only heighten current systems' painful shortcomings.

Historic efforts to cope

The various "coping with information overload" efforts over the years have gone by a number of names including: data pattern processing, information discovery, datamining and knowledge extraction. While the problem to be addressed in this research falls into the broad area of information overload, the investigation was narrowed down to the specific domain area of information systems intrusion detection. Specifically, evidence of malicious cyber behavior (successful or not) represent the “needles” of knowledge that we seek, and given the level of niche successes that various tools demonstrate, we did not feel it was necessary to completely re-invent the mousetrap, but rather believed that the current tools could be teased apart and re-woven together with recent developments in other domains to create a robust, unique and effective problem solving system.

An interdisciplinary approach – combining three fields

Even as datamining has arisen as a direct response to handling the CwIO problem (with intrusion detection as a subfield) two other distinct research arenas have developed over the past

number of years, mostly isolated from one another and not directly related to the CwIO issue: hybrid intelligence, and multi-agent systems. Datamining applications represented the first solid and effective attempts at combating the CwIO issue and, to an extent, traditional and innovative variations of datamining techniques have kept apace of the problem spaces to which they were tailored as long as the complexity of the data remained at a minimum. More often than not though, in most information problem spaces, the solutions have fallen behind the pace of growth.

Hybrid Intelligence "a three stranded cord is not easily broken" [10]

With the advent of multiple, valid technical approaches to solving a given problem, often aimed together at a common problem, the concept of hybrid intelligence sprang up, which Jacobsen describes as "the integration of different learning and adaptation techniques to overcome individual limitations and to achieve synergetic effects through the hybridization or fusion of these techniques." [11]. In our implementation of hybrid intelligence, various machine learning schemes are used together, shoring up the others' weaknesses with their individual strengths.

Multi-agent Systems

For some time now, researchers have been tackling complexity issues across various domains by modelling nature based solutions, employing computational intelligence in multi-agent systems (MAS). In recent years, researchers have seen the benefit of bringing these solution spaces together to attack the CwIO problem [3] [4] [5] [6] [7] [12]. Our work extends this multi-disciplinary research, in the intrusion detection domain, through the development of a multi-agent system which simultaneously operates multiple machine learning/datamining algorithms to identify malicious cyber activity in an offline environment. We posit that this solution yields more insightful results than many current systems, through the folding together of several already successful and proven methodologies in a novel way.

MAS and Datamining combined

Researchers have been discussing the cross breeding of datamining and multi-agent systems for some time, [13] [14] [15] [16] [17] [18] realizing that benefits experienced by other domains can also be had in the intrusion detection arena. For instance, in the knowledge discovery process described by Abraham et al. [3], and outlined in greater detail below, steps 3 (preprocessing) and 4 (data reduction and projection) are widely known to be the most labor intensive [19] [20]; appropriately crafted intelligent agents can do much of this tedious and resource-intensive work. Additionally, in a real world environment, agents work together autonomously 24 hours a day, seven days a week to implement the policies of their crafters. Given the nature of today's cyber battlefield, this level of attentiveness is a requirement rather than a luxury, but an expensive one if handled by human beings. Finally, the modular nature of the agent based architecture easily allows for adaptation and expansion to include altered, new or additional datamining algorithms.

In this work we document the design, construction and demonstration of our SPADE [21] based hybrid intelligence, multi-agent system for mining information assurance data, which integrates multiple learning techniques into one system.

B. Objectives along the path of this research

At the outset of this research a number of milestones to achieve along the way were established, to include the construction and configuration of a lab, generating and mining data suitable to our chosen domain, the comparison and contrasting of results, and then the documenting and fitting of those processes into a hybrid intelligence/multi-agent system. These are briefly touched upon here and explained in more detail in the Approach section.

1) ***Build a lab***

A lab environment was designed and built wherein test data sets could be generated and datamining applications could be tested. The network included a variety of hosts and devices to include: several Linux and Windows hosts, a switch and a hub. The operating systems and their level of patching varied depending on whether or not the machine was to serve as a "victim" or an "attacker" and what services or applications were to be exploited. The "attacker" hosts were loaded with Backtrack (now Kali [22]) or Ubuntu [23] Linux with nmap [24].

2) ***Generate and collect suitable data***

The structure of the lab network and its devices enabled us run exploits from an "attack" host against a "victim" host, allowing for the generation of PCAPs (Packet Capture File [25]) seeded with data of our own choosing. The hub (as opposed to a switch) permitted a host running Wireshark [26] unfettered access to all traffic between hosts communicating on the network. This enabled that host to capture the exploits, as well as both horizontal and vertical scans which were initiated across the network, simulating real attacks, thereby generating the necessary "seeded" data.

Why Intrusion Detection Data

There are several reasons we chose the intrusion detection domain for our research. For one, we hoped that this endeavor would result in some practical applications of datamining in a community full of data but short on the ability to mine it, due to the several well-known and painful shortcomings as outlined later in this paper. Additionally, the measures of success or failure are easily quantified: one finds the intruder and plugs the hole, or suffers the consequences [27]. Also, this arena provides an excellent backdrop against which we could test our greater hypothesis, which is that multiple, and different types of datamining algorithms

working in parallel, running in a multi-agent system, would yield much more knowledge from the data, or would cope better with information overload than would any single algorithm on its own. The research was limited to the examination of offline data.

3) Following a small rabbit trail – an unplanned objective

Wrong assumptions at the outset

It became evident that a side trip in our research would need to be taken upon arriving at Abraham's datamining steps 3 and 4a (data cleaning, preprocessing, and finding useful features to represent the data). At the outset of our endeavor, our expectations were that various tools would already be available for the basic data file conversions required in these steps, however, we were unable to find a suitable tool to convert an entire PCAP to ARFF (Attribute-Relation File Format).

Before deciding to develop our own converter application, several other tools oriented toward similar efforts were discovered and tested for their efficacy, but ultimately, were found lacking for our purposes. One such tool is tcp2d, which is a "tcp stream reassembler" [28] that allows one to "mash and classify packets and create the kinds of data sets" [28] its author desired for testing in Weka [29]. It is a very useful utility and excels in what it does. A second tool is fullstats; this is a "utility used to create the full 266 features as defined by Moore for Traffic Classification," which discards "confidential information from the flows" [30]. These two tools both pare down and reformat the data contained in a PCAP, making them somewhat useful for bridging the gap between PCAPs and Weka (ARFF). However, while these tools are useful in accomplishing the aims for which they were created, they did not fulfill our need for converting the entirety of the PCAP into ARFF. Thus, we had to develop our own application to accomplish this.

Pdml2arff.py – On the trail again

Although PCAPs are easily generated and readily available to any network administrator, and the download and installation of Weka lies within easy grasp of the same for datamining tasks, getting data from the one application to the other is not a simple job. While Wireshark and tshark export PCAP to csv format, which can be ingested by Weka, by default they render only a very few of the thousands of potential fields. A comprehensive search of the literature revealed that there were no readily available tools to convert the entirety of a PCAP file into ARFF format for Weka, essentially locking away the storehouses of information in PCAPs from everyday information security professionals wishing to use those two applications in concert. To overcome this issue, we wrote the pdml2arff.py application, a PCAP to ARFF converter which ultimately defines one step in a clearer path for the layman network/security administrator to mine their own data, by adding a previously non-existent (and very necessary) tool to their kit. Our pdml2arff.py application bridges the gap between two open source products and their output and input data files: Wireshark for capturing network data in PCAP format, and Weka as a datamining application which ingests ARFF files or connects to a database. It is currently posted at Github [31] and is freely available for downloading.

Several attempts were made to write this program in other languages/scripting languages before finally deciding upon the use of Python. In our initial searches we found a very old script written in zshell, which we tried to adapt, but could not find the all of the libraries and dependencies which it required in order to run. Bash was tried, but was just simply too limited in many ways. Perl would probably have worked but we settled on Python in the end because of Python's strong text/xml parsing libraries, object oriented nature, and suitability for the task. Harrington weighs in on this choice as well:

Python is a great language for machine learning for a large number of reasons. First, Python has clear syntax. Second, it makes text manipulation extremely easy. A large number of people and organizations use Python, so there's ample development and documentation. [32]

4) ***Mine the data***

The PCAPs were preprocessed using a number of different conventional tools in addition to tools and scripts which we created along the way. The processes were painstakingly documented and automated as much as possible, in a fashion that would generate well-defined, repeatable and measurable results. From this data standardized metrics were created and captured to enumerate the strengths and weaknesses of various existing algorithms. The details of these processes are spelled out in the Approach section and Appendices.

5) ***Create HI/MAS***

A hybrid intelligence/multi-agent system (HI/MAS) based on the SPADE framework was designed and coded to automate the processes from the previous milestone and to investigate whether or not this approach to offline datamining would produce measurably superior results to conventional, non HI/MAS approaches.

C. Research Questions and Measures of Success

The research questions we sought to answer in this work are as follows:

1) ***Can datamining accuracy be improved by hybridizing various traditional approaches?***

Ultimate success hinged upon a HI/MAS producing superior results over a non-HI/MAS solution, specifically focusing on the hybrid intelligence aspect. It was hypothesized that, at the very least, higher accuracy from the HI/MAS solution would be achieved. Figure 1 provides a matrix wherein test results could be visualized.

2) ***Can benefits be realized from implementing the processes into a multiagent system?***

Given the performance hit against the datamining system's speed as discussed below, if a monolithic system can be recast into a new paradigm (multiagent system), it is quite possible that overall speed may increase. In the multiagent model, tasks are broken down and parceled out to simplified, speedy agents, opening the door for spreading the processes across distributed systems; offloading processing, memory and other resources.

3) ***Can datamining speed be increased by hybridizing various traditional approaches?***

Without doing much research one can safely postulate that adding further processing and resource requirements for more algorithms on top of a potentially already sluggish system will probably not increase the speed. While the answer to this should be fairly obvious, it was posited for the sake of completeness for evaluating all of the pros and cons.

Figure 1. Success Matrix

D. Chapter Summary

In this chapter we roughed out the size, scope and ubiquitous nature of the "coping with information overload" problem. We outlined the historical timeline of solutions to address this moving target and discussed the need for even more innovative solutions in the future to successfully grapple with the issue. We briefly outlined the objectives of this research, which is to implement a novel, cross domain solution to address the coping with information overload problem, and then touched upon high level ways to see if our solution is effective.

E. The remainder of this paper

The remainder of this dissertation is organized as follows:

Section II, the Literature Review dives into the various topics pertinent to this research, fleshing out datamining concepts and approaches, and presenting various approaches to addressing the intrusion detection solution. We also explore the reasons for the disconnect between the two fields of datamining and intrusion detection for users at certain levels, as well as ways to shrink that gap, hopefully empowering every day professionals with tools to better secure their own networks. This section goes on to talk about multi-agent systems and finishes up with a brief enumeration of the tools one would need in order to duplicate our research.

Section III presents the Approach, where we detail the repeatable steps involved in arriving at our solution. Some of the items included here are: data collection, tool acquisition and selection, feature extraction, and running the data through Weka [29] and its various algorithms. This section is wrapped up with a look at some of the factors to consider in designing and building a hybrid intelligence system in a multi-agent framework and outline our own approach to doing so.

Section IV presents the results from the steps performed in the Approach, with minimal discussion, and section V closely analyzes the results, drawing conclusions within the framework of our thesis and expected results.

Finally, Section VI presents a glimpse down the path, forward to future work.

II. LITERATURE REVIEW

Covered in this chapter…

- Datamining
- Intrusion Detection Systems
- The gap between domains
- Multi-agent systems
- Tools to repeat this research

A. A survey of datamining

The end goal of datamining is to find all interesting patterns, and only the interesting patterns, hidden in a set of data sources [33]. There are two datamining models to accomplish this: predictive (e.g. classifiers) and descriptive (e.g. clusterers and associators). Regardless of type, all algorithms do the following:

- Attempt to create models which fit the data
- Allow these models to be compared to preferences in order to determine which model or models are the better ones
- Allow for some sort of search technique. [33]

Each of the types of datamining algorithms described below function differently, bringing along their own benefits and drawbacks. Through the review of previous work where datamining has been applied to the intrusion detection domain [19] [34] [35] [36] [37], and with an understanding of the strengths and weaknesses of each type of algorithm, test scenarios which play to the strengths, and expose the weaknesses of each algorithm were crafted for this research. Furthermore, the examples herein only begin to scratch the surface of what nuggets of knowledge

any given algorithm can lift out of a pile of data, all the more so using multiple algorithms in concert on the same data.

1) *Datamining, by any other name*

Listed below are a number of definitions for "Datamining" culled from the World Wide Web, and as viewed from a variety of different fields:

• "Obtaining information about customers or groups of customers from a data warehouse for marketing or other purposes." [38]

• "Datamining is derogatory. It means sorting through a huge volume of data, extracting decision rules that seem to favor one team over another, but without regard to whether or not there is any cause-and-effect relationship. Datamining is the sports-betting equivalent of sitting a huge number of monkeys down at keyboards, and then reporting on the monkeys who happened to type actual words." [39]

• "...analyzing large amounts of data in search of previously undiscovered business patterns." [40]

• "...look for hidden patterns in data - for example, identifying customers with common interests." [41]

Among the many more generic definitions, these stood out for their industry specific bent. For the purposes of this paper, all of the definitions above loosely apply, however, the most useful one comes from R. L. Grossman: Datamining is "concerned with uncovering patterns, associations, changes, anomalies, and statistically significant structures and events in data." [42]

2) *Knowledge Discovery in Data Bases: By the numbers*

The various historical efforts focused on "coping with information overload" have gone by a number of names including: data pattern processing, information discovery, datamining and

knowledge extraction. Abraham et al. [3] have broken down the knowledge discovery process into a number of steps and sub-steps which are enumerated below and shown in a graphic fashion in Figure 2.

Abraham's steps enumerated

1. Developing and understanding the application domain, the relevant prior knowledge, and identifying the goal of the "Knowledge Discovery in Databases" process.

2. Creating target data set.

3. Data cleaning and preprocessing: basic operations such as the removal of noise, handling missing data fields.

4. Data reduction and projection:

a) Finding useful features to represent the data depending on the goal of the task.

b) Using dimensionality reduction or transformation methods to reduce the effective number of variables under consideration or to find invariant representation of data

5. Matching the goals of the "Knowledge Discovery in Databases" process to a particular datamining method

After spelling out the steps in the process applicable to any datamining endeavor, Abraham et al. [3] go on to enumerate various datamining methods such as: clustering, summation, dependency modeling, regression, classification and change in deviation detection. In the section below, some of these datamining techniques are examined in a bit more detail. It is within the framework of these steps that we tackled the coping with information overload problem as outlined above.

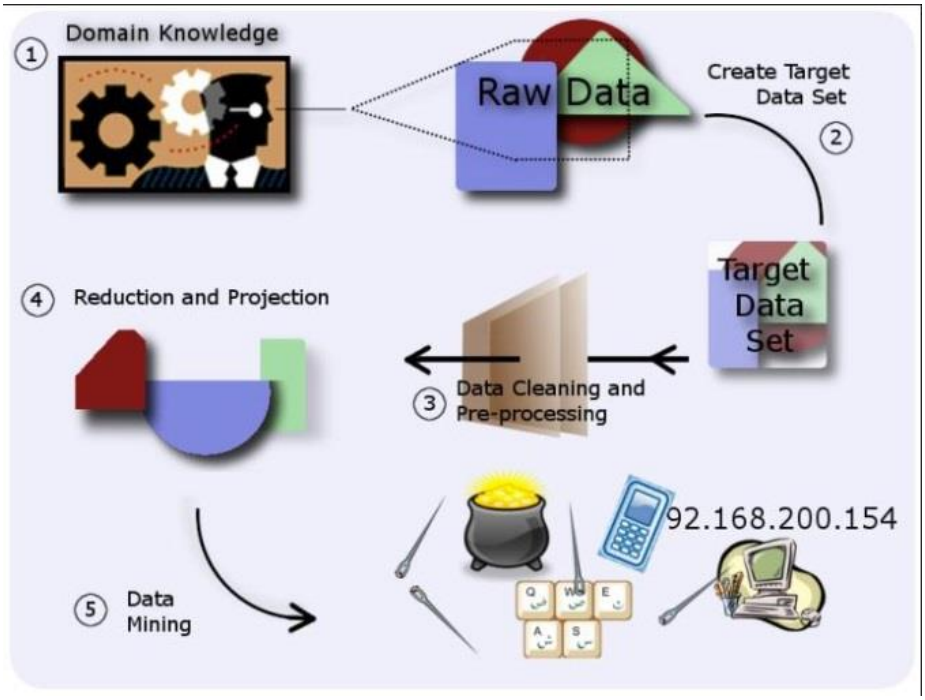

Figure 2: Steps of the knowledge discovery process, adapted from [3]

3) *Traditional Datamining Methodologies*

The following section outlines various traditional datamining methodologies such as classification, clustering, and association, as well as various implementations of each type.

Classification

Classifiers explained and some examples

Classifiers are given a set of training data in which the class attribute that one wishes to classify is already labeled. Additionally, it is hoped and expected that a subject matter expert applied the labels and that they are all accurate. The algorithm examines this pre-labeled data and develops a set of rules which can be visualized as a decision tree. The rules produced from this process should be sufficiently descriptive of the training data, but not so well fitted in their descriptions that they cannot adapt to previously unseen data. This rule-set can then be applied to unlabeled data (of identical structure) which then should be able to predict into which classes the

unlabeled data will fall. [43] Classifiers are usually good at identifying what they have trained themselves to recognize, but fall short when presented with new patterns for which they have no rules, a situation which occurs more frequently than desired in the dynamic world of network traffic. [44] In the world of intrusion detection, this translates into good results when looking for known malicious behavior, but bad results when facing something previously unseen, like a zero-day exploit.

In general, classification techniques include: decision tree based methods, rule based methods, memory-based reasoning, neural networks, support vector machines, Naive Bayes and Bayesian belief networks. Some examples of these are enumerated below.

ID3, C4.5 & C5.0 CLASSIFIER ALGORITHMS

The ID3, C4.5 and C5.0 algorithms are the creations of Ross Quinlan [45]. His C5.0 algorithm, which has a number of improvements over the earlier ones, has been wrapped into commercial packages available for Linux/Unix and Windows.

ID3

The Iterative Dichotomiser 3 algorithm, developed by Ross Quinlan generates a decision tree through a three step process [46].

- Iterate through attributes in a test set of data, and count their entropy (i.e. "If all instances in C are positive, then create YES node and halt. If all instances in C are negative, create a NO node and halt." [46])
- Find (choose) an attribute which generated neither a "yes" nor "no" node, indicating maximum entropy
- Generate a new node containing that attribute and iterate again

C4.5

The C4.5 algorithm, called a statistical classifier, generates decision trees which in turn are used for classification. It was created to overcome shortfalls in the ID3 algorithm, specifically in terms of choosing the best (most normalized) attribute out of the decision tree, for creating subsets.

Improvements that C4.5 brings over ID3 include:

- Avoiding overfitting the data
- Determining how deeply to grow a decision tree.
- Reduced error pruning.
- Rule post-pruning.
- Handling continuous attributes. (e.g., temperature)
- Choosing an appropriate attribute selection measure.
- Handling training data with missing attribute values.
- Handling attributes with differing costs.
- Improving computational efficiency. [45]

C5.0

C5.0 is mostly a recapitulation of C4.5 with the following list of improvements.

- several orders of magnitude faster in its execution
- uses memory more efficiently
- produces results that are just as good while using smaller decision trees
- supports boosting which creates better and more accurate trees
- allows for weighting of different attributes and misclassification types
- reduces noise through "winnowing"

(adapted from [45])

Ensemble Methods

One effort gaining ground towards the end of integrating multiple machine learning approaches is found in the use of "ensemble methods." Implementations of ensemble methods have demonstrated the ability to "increase predictive performance over a single model." [20] Having said that however, these often result in "loss of interpretability" and "results [which] are quite counterintuitive." [20] Nonetheless, after working through these drawbacks, for the steadfast researcher(s), they have proven to be worth the effort in achieving superior results.

Two implementations of ensemble methods are bagging (bootstrap aggregating) and boosting, both of which employ the resampling of one's dataset to yield different models from the same algorithm [47]. In bagging the training set is resampled or broken down into *k* subsets of identical size, each of which are used to train the given classifier, producing *k* models. Each of these models are then aggregated, with each model "casting a vote" on how to classify a training instance, with none receiving preferential treatment over another. Boosting, on the other hand, creates models in an iterative fashion, each successive model being built upon the foundations of the previous model, with weights being assigned to instances which were previously incorrectly classified. Both bagging and boosting can be used for classifying (voting as described above) or for numeric prediction, in which case averaging is used instead of "voting."

The third type of ensemble method is called *stacked generalization* or *stacking* for short [20]. This method differs from the previous two in that it does not slice and dice the data in various ways to train and retrain the same classifying algorithm, but rather, through the use of any of a variety of weighting and voting schemes will combine the results of several different classifer algorithms to yield a result.

WHY ENSEMBLE METHODS DO NOT APPLY TO THIS RESEARCH

At first blush ensemble methods may appear to obviate the need for our research in that they combine the results of multiple inputs to yield an enhanced result. However ensemble methods only combine the results of models which are of the same type, i.e. classifiers or numeric prediction. So while retaining the allegorical concept of garnering input from multiple counselors, we have set aside ensemble methods for the time being, since they are an aggregation of one type of algorithm. They are more of a sub-set of our postulation that input from the other *types* of algorithms will yield even greater insights into information assurance data. In other words, while recognizing their benefits within the realm of classification and numeric prediction algorithms, we take ensemble methods a step (or more) further: integrating results from the various different algorithm classes (classifiers, clusterers, associators), not just tweaking what is being fed into a classifier. Certainly ensemble methods can be folded into the mix in future research, and will probably be helpful, but their use is not necessary to support our current efforts.

Classifiers applied to intrusion detection

Research abounds regarding the practical applications and benefits of classifiers for intrusion detection, for instance: decision trees [48], Hidden Naïve Bayes [49], eXtended classifier systems [34], multiLayer perceptron (MLP), linear classifier, Naïve Bayes classifier, gaussian mixture model (GMM), and support vector machines (SVMs) [50].

Additionally, classifiers provide the foundation for virtually all signature based intrusion detection systems. In these systems, signatures for interesting or undesirable behavior have been previously crafted and are fed into the classification system which searches for a match, which it then flags. As noted above, this technique proves very useful against clearly defined and previously seen attacks, but cannot identify anything it has not already been trained to see.

Clustering

Clustering explained and some examples

Clustering algorithms process unlabeled data and discern natural groupings therein [43] [20], they then put the elements of data into related groups or clusters which have not been pre-defined. Oberst explains some of the benefits of clustering as follows:

> Pattern recognition uses clustering to filter out faces from pictures or video feeds. In medicine clusters represent different types of tissue and blood in three dimensional images, for example to help diagnose tumors. Biologists find functionally related proteins or genes that are coregulated by scanning large DNA sequences. [51]

Clustering algorithms measure similarity in data and partition it according to the discovered, similar datapoints. Original clustering algorithms fell into one of two clusters, if you will, those being partitional and the other, hierarchical. In general "partitional clustering algorithms attempt to determine k partitions that optimize a certain criterion function." [52] The most often used criterion being a square error one. This works well enough when the differences between clusters of data are well defined and the clusters are roughly the same size. However, when the size or geometries of the clusters vary, the square-error methods most often employed work against themselves, splitting the clusters for the wrong reasons (i.e. size and geometry, rather than logical or topical differences in data).

Hierarchical clustering uses an agglomerative algorithm. This type of clustering takes disjointed sets of clusters and successively merges or nests them "until the number of clusters reduces to k. At each step, the pair of clusters merged are the ones between which the distance is

the minimum." [52] In a nutshell, clustering tries to group data into sets such that intra-cluster similarities are maximized, while inter-cluster similarities are minimized. [53]

K-MEANS

The K-means unsupervised clustering learning algorithm is attributed to J. B. MacQueen, laid out in his 1967 paper [54] and has withstood the test of time. It has, understandably however, undergone a number of revisions in the intervening decades since its advent. In an effort to solve well-known clustering problems, the algorithm (as outlined in Figure 3) takes a predetermined set of points (centroids or "K") around which the clusters will eventually form, and emplaces them in some fashion (either randomly or not) into the data set. From this point, a variety of sub-algorithms come into play, each working any of a number of different ploys to most efficiently carve up the data (into clusters) which produce the most desirable clusters at the end of the exercise. This paper will examine three of the more interesting K-means implementations: Lloyd's k-means clustering, Progressive Greedy k-means clustering, and the K-Medoid algorithm.

LLOYD'S K-MEANS CLUSTERING

Lloyd's algorithm separates the data within a partition into a fixed amount (*k)* of clusters, where each center also acts as a representative. As with the original k-means implementation, the algorithm cycles through the data, resetting the centers and reassigning the points as necessary, to the closest new center. This process continues until equilibrium is established, i.e. the centers no longer move. Wilkin et al. observe the interesting fact that "the Lloyd's algorithm often converges to a local minimum of the squared error distortion rather than the global minimum." [55]

At a high level, the K-means algorithm executes 4 steps to process data, as shown in Figure 3:

1. User is prompted for an initial number of clusters or seeds (k). In Figure 3 below, k = 5.

2. The algorithm randomly places those seeds as “k” cluster Centers
3. The data points discover which is the nearest Center.
4. The Center then identifies which one of its points is the centroid.

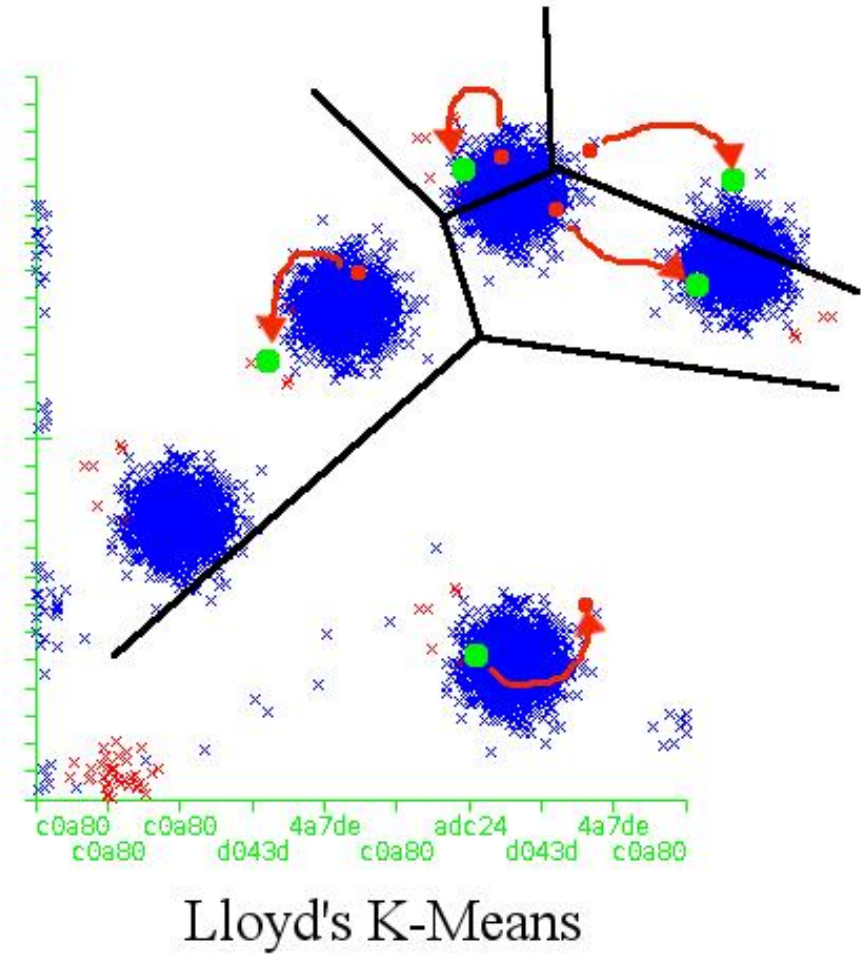

Figure 3. Lloyd’s K-means, adapted from [56]

PROGRESSIVE GREEDY K-MEANS CLUSTERING ALGORITHM [55]

While this algorithm bears striking similarity to Lloyd's algorithm by searching for the best center of gravity to which each point should belong, it differs in the way it assigns each point throughout the iterations. Whereas Lloyd acts upon each point in every iteration, the Progressive

Greedy algorithm calculates the cost (based on a Euclidean distance) of every point and then determines which point would most benefit from being moved into another cluster. Each Progressive Greedy iteration plays out as follows:

Step one - for each point, calculate and assign to each point, the cost of moving it into another cluster center as well as the cost to let it remain where it currently lies.

Step two - If there is one clear winner from the exercise above, go ahead and move it. If there is a tie between two or more points, pick the one which will realize the greatest improvement from having been moved.

Step three - When nothing else can be done, stop iterating.

As a point of interest, for their purposes, researchers in [55] found that Lloyd's worked better for their datasets which contained larger amounts of numerical data. This may not be the case if the data was not numerical, not so large, or a mixture of numerical and categorical data.

K-MEDOID ALGORITHM

K-medoid is another partitioning, clustering algorithm and a variation on the k-means approach. The two most prominent variations of the k-medoid approach are the PAM (Partitioning Around Medoids) and CLARA (Clustering LARge Applications). The K-medoid approaches apportion their clusters through the use of medoids rather than centroids. The medoid is another attempt at locating the most central point in the cluster [57]. As useful as PAM is, it runs into computational complexity issues. CLARA was developed in an effort to get around these.

Instead of finding representative objects for the whole data set, CLARA reduces the complexity by drawing multiple samples of the objects and applying PAM on

each sample. The final medoids are obtained from the best result of these multiple draws. [58]

COBWEB

The Cobweb algorithm is a hierarchical clustering algorithm which instantiates new clusters incrementally instead of starting with a pre-determined number of seeded clusters. The algorithm commences with the formation of 2 clusters and then recursively decides whether to split them into more clusters or join them together. It builds this hierarchical structure "by 'agglomeration' – that is starting with individual instances and successively joining them up into clusters." [20] More specifically, when handed a new instance in the data, the algorithm does one of 4 things:

- places the instance in an already extant cluster

- creates a new cluster in which to place it

- merges two extant cluster and places the new instance in the hierarchy [59]

- splits an existing cluster in two and puts the instance in the resulting hierarchy

Given its hierarchical nature, vs. pre-seeding with clusters, it was determined that Cobweb would be a good second algorithm to use for this research as a foil to compare against the Simple K-Means runs.

ROCK ALGORITHM

The Stanford University and Bell Labs researchers [60] who developed the ROCK algorithm point out a number of shortcomings of earlier clustering algorithms such as the inability of centroid based clustering to effectively handle data sets with categorical attributes (vs. numerical attributes where they perform satisfactorily). They go on to point out that previous hierarchical algorithms also fail to adequately address data better characterized by categorical attributes,

positing that "distances between centroids of clusters is a poor estimate of the similarity between them." [60]. To overcome some of the perceived shortcomings they introduce several new concepts in their approach. Typical partitional and hierarchical clustering algorithms are well suited for numerical and binary data, clustering data points based on a measure of distance (Jaccard coefficient), but these methods are not well suited for the clustering of categorical data. The researchers developed their ROCK algorithm to better handle data which can be broken down into categories, by clustering on links between data points such as mapping grocery shoppers trends i.e. diaper, milk and toy buyers vs. the wine, TV dinner and imported chocolate consumers. [60]

BIRCH ALGORITHM

"BIRCH uses a hierarchical data structure called Clustering Feature tree (CF tree) for partitioning the incoming data points in an incremental and dynamic way." [51] BIRCH applies a four step process as follows:

• Linearly scan all data points and insert them in the CF tree.

• Condense the CF tree to a desirable size depending on the clustering algorithm employed in step three. This can involve removing outliers and further merging of clusters.

• Employ a global clustering algorithm using the CF tree's leaves as input. The Clustering Features allow for effective distance metrics. The authors [51] describe different metrics with time complexity of O(N2). This is feasible because the CF tree is very densely compressed at this point.

• Optionally refine the output of step three. All clusters are now stored in memory. If desired the actual data points can be associated with the generated clusters by reading all points from disk again [51].

CURE ALGORITHM

CURE (Clustering Using Representatives), an algorithm developed by Stanford University and Bell Laboratories researchers [52], offers a different slant on hierarchical clustering which claims to adopt "a middle ground between centroid-based and all point extremes" [52] based approaches. CURE starts with a number of points which are "well-scattered" across the data space, and then shrunk towards the centroid of the cluster. These newly shrunk points are now the representatives of the new cluster. The process then calls for merging the newly minted clusters which have the closest pair of representative points. According to the authors this approach overcomes shortfalls in both the all-points and centroid-based approaches by allowing for better, or more fully correct identification of clusters and allows for varying geometry not seen in the other approaches (i.e. elongated vs. only spherical). They posit that these allow for a more accurate reflection of the data space.

CACTUS ALGORITHM

Researchers developing the CACTUS (**Ca**tegorical **C**lus**T**ering **U**sing **S**ummaries) algorithm [61] categorize it as a fast summarization based algorithm for the clustering of categorical data. They implement three phases: *summarization, clustering* and *validation*. Rather than digging deeply down into the data, they postulate that, for their purposes, "summary information constructed from the dataset is sufficient for discovering well-defined clusters" [61]; furthermore, the properties they glean fit into memory, using only one scan of the dataset efficaciously provides construction and generates relatively speedy results.

The researchers point out that "categorical attributes usually have small domains" [61], often less than 100 and in rare instances, less than 1000, and this is one of the linchpins of their approach, or claims to speediness, so that all of the work fits into main memory. In the

summarization phase of the CACTUS, their approach computes the inter-attribute and intra-attribute summaries.

The researchers implement a three step process: a two-step clustering phase wherein they compute candidate clusters in the data using attribute summaries they derived in the previous phase. Step one is comprised of the computing of all cluster projections retrieved from attribute analysis on the data. In the second step they "synthesize, in a level-wise manner, candidate clusters on sets of attributes from the cluster-projections on individual attributes." [61] In other words, after drawing out candidate clusters from a pair of attributes, they extend that pair into a new set of three attributes, repeating as often as necessary. In the third step, or validation phase, they take the set of candidate clusters and extract the real clusters. They implement a variety of gates through which a candidate cluster must successfully pass in order to become an actual cluster. They indicate that their validation phase is robust and proves their theorem.

K-HISTOGRAMS

The *k*-histogram algorithm is an extension of the *k*-means algorithm into the categorical domain. It replaces the means of clusters with histograms and will dynamically update histograms during the clustering process. Chinese researchers [53] indicate that their implementation of this type of algorithm produces clustering results superior to the *k*-modes algorithm (a k-means variant to account for categorical objects using dissimilarity measures) [62], which is the one most closely vying against them for the next level.

The *k*-histogram researchers observe that while previous clustering algorithms may excel at parsing out numerical data, their performance and results are questionable or not so useful when dealing with categorical data, i.e. "shape" which could have the various values including: circle, rectangle, ellipse, etc. They also point out the shortcomings of previous clustering algorithms'

(STIRR, CACTUS, ROCK, Squeezer, and *k*-modes) attempts at addressing the "categorical problem." They replace k-modes means of clusters with histograms, at the same time updating the histograms dynamically during the clustering process, using the histogram data to describe the categorical data cluster.

INCREMENTAL MULTI-CENTROID, MULTI-RUN SAMPLING SCHEME (S) [58]

The IMCMRS was developed as an improvement over the MCMRS working with k-medoids-based clustering and claims to render an 80% improvement in computation time. One interesting thing to note from this work is the enormous performance gain reaped from infusing a well-established technology with another technology.

Clustering applied to intrusion detection

Research has also been conducted to show the effectiveness of clustering algorithms [44] [63] [64] in the realm of network intrusion. In this domain, among other things, clustering algorithms lend themselves well to identifying hosts which communicate with each other, regardless of the legitimacy of the exchanges. To take advantage of this inherent strength, data can be processed in such a way as to maximize exposure of interactions of a host with other hosts. Information security professionals are very interested in certain types of communications, especially scanning. Specifically, there are two types of scanning behavior, *vertical* and *horizontal* [65]. When one host scans one other host on a wide range of ports, this is a *vertical* scan. Typically the malefactor is looking for any of a number of open ports through which it can apply any of a number of exploits which would make it possible to infiltrate the one specifically targeted host. A horizontal scan, on the other hand is when one host scans multiple hosts on a network segment, usually looking for one, or a small set of open ports in order to apply a specific exploit, in which case any vulnerable host will do. To ascertain whether or not the communication is legitimate or malicious

requires further inspection, either in an automated fashion or with the insertion of a human subject matter expert into the loop.

On the other hand, clustering algorithms are not "trained," they are unsupervised and cannot be taught what phenomena would be considered "good" or "bad" from a security standpoint. After performing their clustering, the results need to be examined by a human who is knowledgeable in the domain or by another learning algorithm in order to interpret the results.

Associators

Association explained and some examples

Association rule mining algorithms discover correlative but not necessarily intuitive or expected relationships between items in large data sets [66] [67].

> Association algorithms find correlations between different attributes in a dataset. The most common application of this kind of algorithm is for creating association rules, which can be used in a market basket analysis. [68]

In other words, instead of breaking sets of data down into more distinct groups, association algorithms attempt to find relationships in (possibly disparate) types of data.

The "market basket" analogy is frequently used to explain the benefits of association rule mining,

> According to datamining urban legend, a study of customers' purchase behavior in a supermarket found that men often purchased beer and diapers together. After making this discovery, the managers strategically placed beer and diapers closer together on the shelves and saw a dramatic increase in sales. [69]

Briefly, this process is accomplished in two steps: first, by the discovery of items (antecedents) which appear in the data and grouping those together (consequent) into *itemsets*, and second, by generating rules which have a high confidence. The two step process put another way is:

> 1. Find all frequent itemsets: By definition, each of these itemsets will occur at least as frequently as a predetermined minimum support count, min_sup.
> 2. Generate strong association rules from the frequent itemsets: by definition, these rules must satisfy minimum support and minimum confidence. [70]

For an association rule to be considered useful at all, any given *itemset* must be found within the entire data set a user-determined percentage of times; this is called the *support* or *minimum support* for the rule. Additionally the rule must meet a certainty, or *confidence* threshold, which is determined by requiring the items to be found together a minimum percentage of times.

As an example, the following rule indicates that customers who bought niacin also purchased safflower oil at the same time:

niacin => safflower oil [support = 3%, confidence 40%]

The *support* number for the rule above indicates that niacin was bought with safflower oil in 3% of all the transactions in the data being examined. The *confidence* number indicates that 40% of the times that niacin was purchased; safflower oil was bought along with it. The support and confidence numbers are determined ahead of time and only the rules rising to meet those thresholds are considered interesting. [70]

Examples of association algorithms include: the Apriori Algorithm, Eclat, FP Growth algorithm and others. A brief synopsis and comparison of these are outlined below.

Apriori is the original association algorithm [70] and the discussion above applies most specifically to this algorithm. To help understand improvements upon Apriori, described below, it is useful to mention that it employs a BFS (breadth first search) method and handles data in a horizontal format. Apriori's intensive resource requirements are due in part to this way of formatting.

The FP Growth algorithm also presents data in a horizontal format but overcomes another bottleneck in Apriori which is itemset candidate generation. This algorithm makes two passes over the data and generates what it calls and "FP-Tree" which is a compact data structure. It then extracts all the frequent itemsets from that data structure (tree). The literature indicates that this methodology decreases computing resource requirements but does not offer much improvement in terms of accuracy of results. [71]

Eclat is an acronym which stands for: Equivalence Class Clustering and bottom up Lattice Traversal. This algorithm improves upon Apriori in several ways to include: searching data in a DFS (depth-first search) manner and presenting data in a vertical format. This format is more "normalized" in database terms and speeds up searches as compared to the original Apriori algorithm. [72]

For the purposes of this research, any of these algorithms would have sufficed in that they are all associators. Apriori was chosen because it is the de facto standard for association rules mining and is in Weka.

Association in network intrusion

As described briefly in the Introduction, associators are well suited to act as anomaly detectors, comparing known good traffic to future traffic, looking for anything "out of the ordinary." Association rules are most often used to characterize and establish a baseline understanding of

one's typical network traffic and usage [36] [73]. This "normal usage" profile is then used as a template to detect anomalous, and potentially malicious, behavior in networks [73] [74] [67].

In efforts to create a baseline of network traffic behavior, it is best to give the association algorithm as much data as possible about the network under examination, in order for it to establish frequent itemsets and create useful and accurate rules. Be advised this can be a very resource intensive process. Additionally, while in a real-world situation this should not be an issue, when in a lab setting, while combining PCAPs from different captures on different networks, special care must be taken to ensure that the preprocessed data ends up being structured identically. For instance, account must be taken for cases where attributes from disparate captures (PCAPs) may contain differing protocols, or where the attributes may appear in different orders and may or may not overlap each other.

4) ***A Newer Datamining Methodology***

MapReduce

Google's [75] MapReduce [76] paradigm takes advantage of parallel programming and distributed systems to spread out the task of iterating through massive amounts of data in order to create a map of the data. As simple as this sounds, its advent unlocks the doors onto previously undreamed of insights into massive amounts of data. Briefly, input files are split into X pieces which are then run through a "map" procedure, which sorts and filters the data into logical queues. These queues are then passed to a "reduce" procedure which summarizes the data. A master node oversees the operations of worker nodes which accomplish these map and reduce processes and reports to a user level application. Figure 4 shows a high level, visual breakdown of the overall MapReduce process.

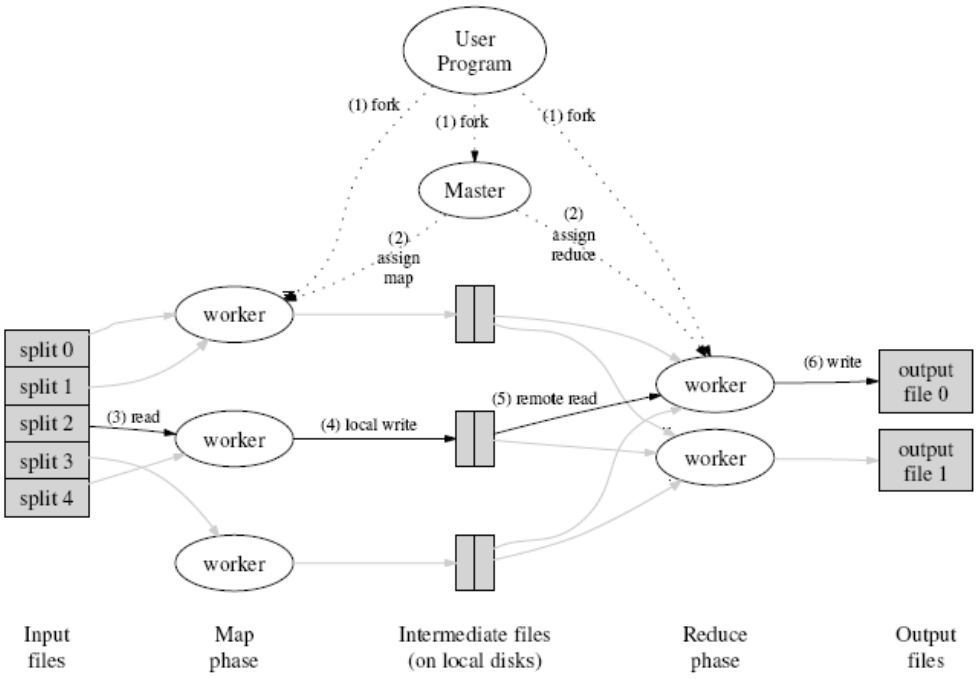

Figure 4. Overall flow of a MapReduce operation [77]

Amazon [78], in their continuing efforts to expand their business by exploiting their current infrastructure and offering more services to their growing customer base, has instantiated a MapReduce on demand service which they call Amazon Elastic MapReduce. They describe it as follows:

> Amazon Elastic MapReduce is a web service that enables businesses, researchers, data analysts, and developers to easily and cost-effectively process vast amounts of data. It utilizes a hosted Hadoop framework running on the web-scale infrastructure of Amazon Elastic Compute Cloud (Amazon EC2) and Amazon Simple Storage Service (Amazon S3). [79]

Amazon employs Yahoo!'s [80] implementation of MapReduce - Hadoop! [81], which Yahoo! donated to the open source community under the auspices of Apache.org [82].

MapReduce Detractors

Not everyone shares in the hype surrounding MapReduce. These detractors mostly come from within the hardcore Relational Database Management Systems proponents. They claim that this new methodology is:

• A giant step backward in the programming paradigm for large-scale data intensive applications

• A sub-optimal implementation, in that it uses brute force instead of indexing

• Not novel at all -- it represents a specific implementation of well-known techniques developed nearly 25 years ago

• Missing most of the features that are routinely included in current DBMS

• Incompatible with all of the tools DBMS users have come to depend on [83]

Stonebraker [83] believes that because MapReduce always performs a full table scan it is a reversion. He posits that it is a poor implementation due to the fact that it has neither indices nor optimization. The third bullet can probably be chalked up to stone throwing. Quite a few well-known techniques do not become efficacious or useful until technology catches up, facilitating their implementation. Additionally, as the folks at Happy Jack Software [84] point out that, while the concept is not new:

> Google made two key observations:
>
> • many practical problems can be expressed as MapReduce problems
>
> • MapReduce can be generalized to process hashtables as well as lists [84]

The features which Stonebraker points out as missing surround ad-hoc and declarative queries. In his last point he decries the lack of tools commonly expected by those working in the data manipulation field; for instance a schema manager and backup utilities [84]. In the end, the

MapReduce's biggest detractors take issue with the accurate but ultimately red-herring observation that it is not an RDBMS. This is good news for proponents of MapReduce who are trying to solve a problem not suited to even today's cutting edge RDBMSes, but instead choose a tool more appropriate for their job of sifting through massive amounts of data.

Interestingly enough, the Stonebraker article was not up very long before it was taken down and replaced with a polite request for the column's readers to back off in their apparently vehement reactions to his post, and a slightly more defensive posture...

> Last week's MapReduce post attracted tens of thousands of readers and generated many comments, almost all of them attacking our critique. Just to let you know, we do not hold a personal grudge against MapReduce. MapReduce did not kill our dog, steal our car, or try and date our daughters.
>
> Our motivations for writing about MapReduce stem from MapReduce being increasingly seen as the most advanced and/or only way to analyze massive datasets. Advocates promote the tool without seemingly paying attention to years of academic and commercial database research and real world use. [83]

As noted in the discussions above concerning other datamining methodologies, different tools excel at different jobs, and the MapReduce paradigm is no exception. While it may suffer from all of the faults outlined by Stonebraker, it has still shown itself to be a promising new way to sift through enormous amounts of data, not only finding needles of knowledge that inquiring minds already know to ask, but also popping out relationships and information that one may not even know hides in the data. Having said that, the employing of it turns out to be quite a significant

undertaking, involving relatively in depth knowledge of either programming in Java or learning a new abstracted language called Pig (if using Hadoop!). It also requires a significant amount of hardware components as well as their setup and configuration. The addition of this algorithm to the cadre would be suitable for a follow on paper and would most likely elicit additional information from the target data sets. However, like ensemble methods, for the amount of extra work, required expertise and knowledge and hardware required, it was deemed that it would not be beneficial, nor necessary to prove the point of this paper.

B. A look at Intrusion Detection Systems (IDS)

It was with good (or maybe naïve) intentions that an open systems approach was followed by the creators of "The Internet." Thanks to this, the world's malefactors were afforded easy access to the many to many networks, and they quickly began plying their age old nefarious practices in this new cyber world. [85] In the due course of the evolution of these networks, intrusion detection systems also cropped up, hoping to catch and even prevent these undesirable behaviors on a network.

1) IDS, IPS, IDPS

A number of tools are available to network security professionals who wish to monitor a network for unauthorized (or any) activity. Among these are IDS (Intrusion Detection Systems), IPS (Intrusion Prevention Systems) and IDPS (Intrusion Detection and Prevention Systems) [86]. Beyond the technical differentiators (discussed below), these systems can be broadly broken down into two groups: commercial and open source offerings. Commercial systems are typically quite expensive [87] and proprietary in how they function but are usually graced with a useful and intuitive user interface, and they generate output suitable for consumption by management level policy makers. On the other hand, while the open source systems are typically very

inexpensive or free, and while their innards are open for inspection and alteration, they generally require much hand crafting and a team of variously disciplined experts to render them effective, from start to finish [88].

2) IDS Basics

Types of IDS

Intrusion detection systems are conceptually divided two different ways, one is by location [89]:

- NIDS - Network Intrusion Detection systems typically have a network port operating in promiscuous mode capturing every packet from the network segment to which they are connected.
- HIDS - Host Intrusion Detection systems are typically a software component installed on an individual host computer.
- DIDS - Distributed Intrusion Detection systems are a network of any combination of either host-based or network-based IDSes which are distributed in one network, or across network segments. The DIDS is differentiated by the presence of a centralized manager to which the distributed nodes report.

The second way of classifying IDSes is by methodology: misuse or anomaly detection. Misuse-based, also known as rule-based IDSes, house signatures of known exploits or other undesirable behavior and compare incoming traffic against them, checking for a match. Anomaly-based systems create a baseline of "normal" traffic on a given network and then identify and flag "anomalous" behavior for inspection. [90]

Advent of IPS

As a hopeful improvement on these systems, *intrusion prevention systems* (IPS) attempt to perform this analysis and provide actionable triggers in real time, interacting with network guards to shutdown perceived unauthorized access. While our efforts, documented herein, focus solely on the offline data problem, future work could certainly bring the functions forward into real time data (IPS), functioning as an integral and enhanced part of an active network defense system.

IDS Drawbacks

Regardless of collection point or methodology, each of these types of IDS may monitor and inspect one of several components of the information system or network traffic stream to include (but not limited to): applications, file systems, log files, host to host(s) communication patterns, network flows and packet inspection.

As a byproduct of doing their job of safeguarding information systems effectively, these systems generate massive amounts of data, often capturing every packet which traverses an internetwork for collation (reconstruction of a communications stream, scattered by the nature of TCP/IP) and correlation (matching upstream and downstream packet flows, or "sides" of a network conversation). After the data have been collated and correlated, the system then begins the laborious task of sifting through it, looking for evidence of illegitimate traffic, employing a variety of techniques, most of them signature based. Intrusion prevention systems attempt to perform this analysis in real time, interacting with network guards to shutdown perceived unauthorized access.

Furthermore, IDSes notoriously produce many false positives [89] [91] requiring human interaction or oversight to weed out the known innocuous events. Additionally, given the unique

nature of every network and host on the network, heuristic, or anomaly based IDSes can take a long time to train.

C. The disconnect between Datamining and Security Administration

1) *Easy to collect data*

Within the information assurance and intrusion detection domain, the conscientious and alert network administrator may suspect that undesirable actors are infiltrating, traversing or exfiltrating data from the domain under their purview. In this scenario, the "needles" of knowledge in the "haystack" of data can consist of data outliers in multiple formats: malformed packets, more than one MAC address showing up as being assigned to a single IP address, oddly placed 'flags' in packets, or other network anomalies [92].

Fortunately, there are a number of standards-based tools which provide comprehensive snapshots of traffic on a network, which can be readily captured at any location in their internetwork and offer a small window of insight into the traffic. Unfortunately, as previously observed, these tools generate copious amounts of data in only a matter of minutes, which is unwieldy and useless without extensive massaging, preprocessing and the proper follow-on tools to decipher it. Furthermore, the most useful tools for identifying and tracking these network anomalies are typically very expensive [87].

2) *Difficult to mine data*

Two fundamental components of an intrusion detection system are the capturing of data relevant to the monitored network, wherein intrusions (or attempted intrusions) are found, and the mining of that data for information identifying the intrusions [87]. The first component lies well within the purview and skill sets of the everyday network administrator or security professional in the form of tools like Wireshark [26] and Snort [93]; the second – data mining,

however, typically does not. For this reason, the solutions are almost always pre-constructed as either a commercial or open source off the shelf package. Also for this reason, we documented and explained the procedures and processes in this research in as much detail as possible, hopefully helping to bridge that divide for the everyday network and security administrator.

Despite the availability and ready acceptance of knowledge discovery through datamining in other fields [94] [95] [96] [97] [98], the information security domain has not been so quick to implement it. While a variety of well-known and widely used anomaly and misuse-based tools exist to examine data for malicious intent (IDSes and IPSes discussed above), other tools with different approaches are available to administrators in the form of datamining software. However, these tools are not typically taken advantage of, and unfortunately, this is with good reason; datamining tools come with a steep learning curve of their own [99]. As Markey points out,

> Historically, datamining techniques have not been adopted in the IT security community. This lack of adoption has been for a number of reasons including the difficulty obtaining the necessary data and a lack of awareness about the techniques. [48]

Suh notes other barriers to entry for datamining into the IT security field:

> Simple, blind application of data-mining algorithms can lead to the discovery of meaningless and useless 'knowledge' from databases. Hence, data-mining systems have to be carefully customized to fit the individual needs of the intended users. [100]

Customization and specialized domain knowledge mean more entries on the cost side of the ledger, often difficult to justify when things are working "well enough" already.

Snort

One example of a widely used, open source, misuse-based intrusion detection tool which works "well enough" is Snort [93]. This particular tool operates much like a datamining classifier, constantly being fed pre-classified signatures of malicious behavior on an ongoing basis as new exploits are discovered and characterized. The program then compares data traversing its interfaces to a list of known exploits, flagging undesirable behavior when it finds a signature match. This tool is mature and well respected for what it does, and has well known benefits in addition to some known-blind spots, such as zero-day attacks for which it would not yet have a signature. One of the points of this research was to demonstrate that, even if a signature-based tool like Snort *did* have zero-day attack signatures, like the blind man who only feels the tusk of an elephant, it would still miss out on other aspects of truth regarding traffic in the network; other aberrant network behaviors that tools based on different methods/algorithms are very likely to pick up.

Using tools like this and others, network administrators and security professionals can easily and readily acquire detailed data regarding activity in their network. However, the lack of having this second skill (datamining and its interpretation) leaves the systems administrator with gigabytes of captured traffic but no real way to make any sense of it. Nevlud [92] outlines three datamining methods for detecting network anomalies which could help the initiate: signature comparison, stateful protocol analysis, and behavioral analysis. Most administrators are left to manually sorting through their large PCAPs and/or lengthy log files, looking for the various pointers indicative of an attack (odd flags, malformed packets, etc.), which can be a time consuming, arduous and a mostly fruitless endeavor.

3) *Closing the gap*

As discussed in previous work [101] [102], datamining software can offer the network administrator insights into their network or hosts through application logs, audit files or network traces (i.e. PCAPs – Packet Captures). One of the ancillary goals of this work was to help further close the gap between these two fields (intrusion detection and datamining) through the exploration and documenting of methods and procedures that are accessible and useful to the everyday enterprise network caretaker. In this work we provide hands on examples so that reasonably knowledgeable administrators can begin to unlock tools which typically lie outside of their knowledge domain.

D. Multi-agent systems

DeLoach, Oyenan, and Matson describe Multi-agent systems (MAS) as using "groups of self-directed agents working together to achieve a common goal." [12] As the complexity of our technological problems has increased scientists and researchers have increasingly turned to nature for inspiration to provide answers. The fields of intrusion detection and datamining are no strangers to complexity and overwhelming amounts of data. As part of the continuing literature review, the section below outlines various solutions already being derived from nature, and look at how they apply to the research solutions proposed in this paper.

1) *Solutions Derived from Biological Behaviors*

Go to the ant, you sluggard;

consider its ways and be wise!

It has no commander,

no overseer or ruler,

yet it stores its provisions in summer

and gathers its food at harvest.

Proverbs 6:6-8 [103]

In the early 1990s researchers began to develop algorithms which mimicked swarming and collective behaviors in order to help solve "well-studied optimization problems like NP-hard problems (Traveling Salesman Problem, Quadratic Assignment Problem, Graph problems), network routing, clustering, datamining, job scheduling etc." [3] Biological organisms with arguably very little computing power for brains, face and successfully solve exceedingly complex problems on a moment to moment basis every day. It stands to reason that we should be able to apply lessons from nature to the solving of our own issues.

2) ***MAS explained: The blind men & the elephant***

As an analogy, a widely circulated fable tells the tale of a number of blind men describing an elephant. One of the men feels the ear and confidently determines that an elephant is flat and leathery, the next man handles the tail pronouncing that an elephant is ropelike, long and skinny, another feels the tusks, and imparts his definitive description, and so on and so forth. The "parable implies that one's subjective experience can be true, but that such experience is inherently limited by its failure to account for other truths or a totality of truth." [104] The moral of this story is no less true in regards to the tools available for enterprise cybersecurity professionals.

3) ***Parable mapped to different datamining algorithms***

Not unlike the parable's various blind men, in the realm of datamining there are numerous methods to parse, quantify, qualify, and elicit patterns from within data, in the ultimate hope of garnering knowledge. These diverse methods are exemplified in the variety of types of algorithms such as classifiers, clusterers, associators and others. Each type of algorithm

approaches data examination differently, providing insights hidden from the other algorithms, yet having blind spots of their own.

4) ***Copying nature***

Technologists frequently turn to examples found in nature in their efforts to design systems which efficiently address complex problems [105]. The development of the multiagent system serves as a prime example of this. Researchers have adapted lessons taken from ant and bee colonies to solve technical problems in the digital domain with excellent results [37] [106]. At their most basic level, multiagent systems consist of computing elements (agents) which exhibit two fundamental characteristics: they are capable of *autonomous actions* and they can *interact* with other agents. [107] Weiss enumerates the following adjectives in describing agents: autonomous, pro-active, ability to perceive reason and act, benevolent, introspective, mobile, rational, reactive [108]. These attributes are part of what set "agents" apart from standalone, stove-piped type software application.

5) ***Platforms***

While a number of standards have been proposed for multi-agent systems [109], the one which has risen to the top is that which is spelled out by the Foundation for Intelligent Physical Agents, or FIPA [110]. The broad categories covered by the FIPA standard are: agent communications, transport, management, abstract architecture and applications.

6) ***Properties of an agent***

Agents can be tasked *directly* or *indirectly*. Direct tasking is characterized by telling an agent what to do, and how to do it, for example, having it execute a program on a schedule. In indirect tasking, an agent is told what to do, but not necessarily how to do it, rather, the agent is given a performance measure to meet. [107] In addition to types of tasking, agents can also perform

different behaviors, which can be run simultaneously if need be. Examples of these behaviors are:

Cyclic and Periodic - useful for performing repetitive tasks

One-shot and Time-out – used to perform casual tasks

Finite State Machine (FSM) – enables the construction of more complex behaviors

Event – enables a response (or responses) by the agent to perceived events [21]

7) ***Collective and swarm behaviors***

The wary diver catches sight of something shadowy, large-ish and predatory looking shimmering in and out of sight in the distance. After a few tense moments pass, the mass approaching more closely, he realizes it is just a school of fish. Who would have guessed? The individual fish in the school or shoal are not generally credited with knowing that if they all swim tightly together "as one," that they will look like a bigger creature possibly discouraging would-be predators. And yet it happens all the time, day and night, throughout our world's oceans. [111]

The individual fish are hardwired to practice certain simple behaviors. The complex patterns which show up in the school or shoal as a result of the collective outworking of these simple behaviors is called "emergent behavior," which is the product, not of an orchestrated or organized effort on behalf of the group of individuals, but rather the acting out of their simple, individual, instinctive rulesets. With that in mind, one could study an individual fish for years, and would never guess that it would "school" or "shoal" when it swam with other fish. Likewise, through the intensive study of a single goose or a single ant, one would never hypothesize that the goose would fly in a "V" formation when migrating with other geese, nor that ants would "queue up" or "form a line" as they act out their simplistic instinctual programming, demonstrating this "emergent behavior."

Romanian and Korean researchers enumerate "the main principles of collective behavior" [3] as such:

• Homogeneity: every bird in flock has the same behavior model. The flock moves without a leader, even though temporary leaders seem to appear.

• Locality: The motion of each bird is only influenced by its nearest flock mates.

Vision is considered to be the most important senses for flock organization.

• Collision Avoidance: avoid with nearby flock mates.

• Velocity Matching: attempt to match velocity with nearby flock mates.

• Flock Centering: attempt to stay close to nearby flock mates [3]

Individuals follow simple local rules which govern how they move and operate within these groups. "Stigmergy" refers to the reactions and interactions of the individual's implementation of these simple rules with their environment including other agents or individuals [112]. Research indicates [111] that there really is no "leader of the flock" or commander, no puppeteer pulling the strings of the bee hive, ant colony or flock of birds. Rather, the combination of stigmergy, a dynamic environment (coral reef up ahead, predator off to the left) and an individual's perturbations in their interactions "drive" or command the flock, cause the collective entity to go to in any of its various free flowing directions.

Couzin et al. [113] further refine the concept by defining 4 "collective dynamical behaviors" including a swarm, torus, dynamic parallel group and highly parallel group. These are illustrated in Figure 5.

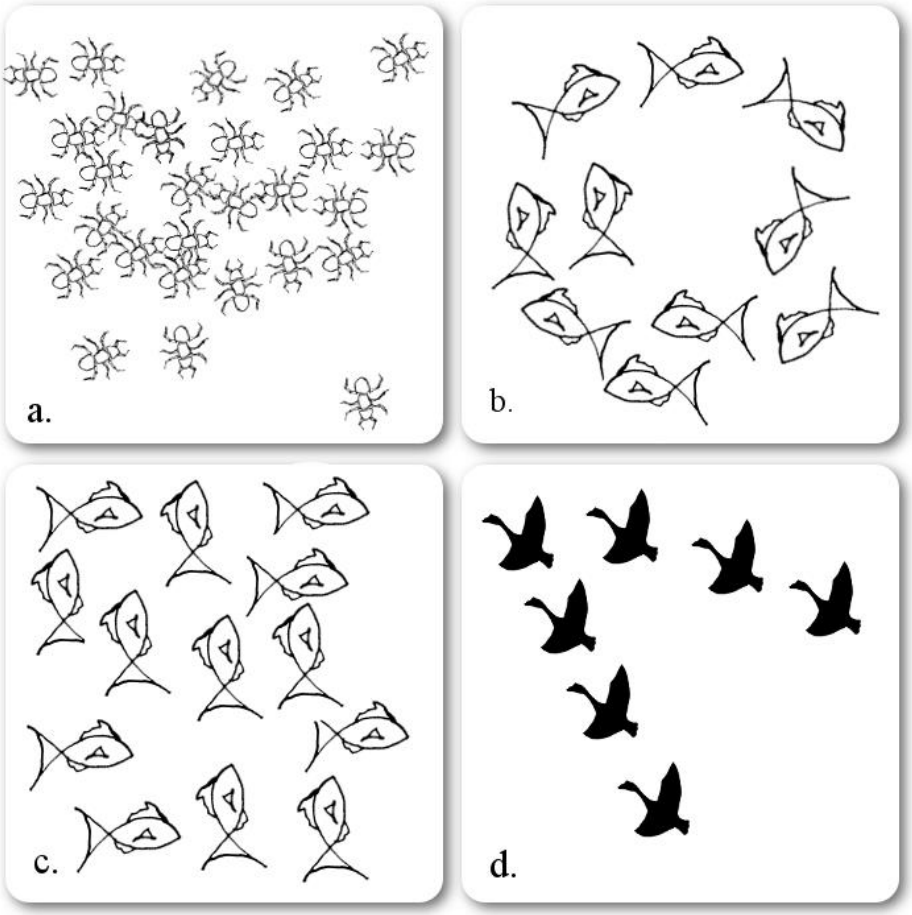

Figure 5. Models of collective behavior. (a) swarm (b) torus (c) dynamic parallel group (d) highly parallel group. Adapted from [113]

8) ***MAS & Hybrid Intelligence: Solutions in practice***

This section describes several recent hybridized systems incorporating various approaches to datamining. No systems were found operating in the intrusion detection domain which incorporated multiple types of datamining algorithms (classifiers, clusterers and associators), or account for human feedback.

TCMMADM

TCMMADM (Traditional Chinese Medicine - Multi Agent Datamining Model) is a multi-agent datamining environment to support knowledge discovery and

modeling of domains with complex issue base on TCM cases. In which different types of agent perform any DM task and integrate partial knowledge into more high level knowledge if it is the necessary. [5]

Zhu et al. [5] seek to combine the best aspects of both traditional datamining techniques (classification, association, and clustering, elucidated earlier) and multi-agent systems techniques. They outline the following advantages lent to their approach by the agent approach:

• Autonomy - Agents can be programmed and let loose in an environment, where they continue to operate without direct outside intervention. They maintain control over their internal state, and make decisions as need be in an effort to accomplish the tasks set before them.

• Socially minded - Agents cooperate with other agents in the Multi-Agent System (MAS)

• Stigmergic - (which the authors categorize as reactive) Agents in a MAS interact dynamically with their environment (hopefully in a timely manner) as it changes.

• Proactive - Agents take initiative to work out their tasks.

Zhu et al. [5] further outline three primitives common to the ideal MAS:

• The MAS framework establishes a set of protocols for agents' communication and interactions

• The MAS framework provides an open and decentralized space within which the agents can effectively work, without requiring a priori knowledge of population size of other agents or their capabilities (providing new agents adhere to the framework's communications standards.)

• The MAS framework allows for heterogeneous, possibly non-similar agent types to work side by side, either together or separately without interfering with one another's tasks and mission.

The TCMMADM researchers continue their approach by enumerating the 3 general types of datamining: association, clustering and classification (fleshed out above), and note more recent forays which add artificial neural networks, genetic algorithms, and support vector machines.

In their system, one finds two types of data sets, physical and perspective, a manager agent (Ama), a cooperator agent (Aco), and miner agents (Ami). The miner agents employ one of three methods: K-means, Apriori, and ID3. Their system handles structured and unstructured data (heterogeneous), which can be physically distributed across multiple systems. Additionally, their system can accommodate knowledge extracted from different perspectives from the same data. They explain that concept in the following manner.

> ...the association rules of consist of items bought together at the same point of time, which can help the management of supermarket decide how to place goods on shelves in order maximize the profit, Furthermore, if analyzing the data by time sequence, it also can get useful knowledge about the trend about what type of goods sale-well in different phase, that is to say, it is quite possible that the same data set is worked on by several different DM algorithms. [5]

Quantitative Association Rule Mining Using a Hybrid PSO/ACO Algorithm

Middle Eastern researchers [6], in their studies on Particle Swarm Optimization (PSO) and Ant Colony Optimization (ACO) point out some of the shortcomings traditional datamining association rules techniques (i.e. loss of knowledge through the discretizing of data into all numerical attributes) and have introduced an "algorithm for mining quantitative association rules using swarm intelligence." [6] Their work demonstrates some of the benefits on offer by recasting traditional data mining into a nature based solutions approach (PSO and ACO). The

approach allows association algorithms to be used outside of their normal constraints which ultimately allows them to mine more of the data, more effectively.

Hybrid Intelligent Systems

The International Journal of Hybrid Intelligent Systems (IJHIS) says the following in describing "hybrid intelligent systems":

> Hybridization of intelligent systems is a promising research field of computational intelligence focusing on synergistic combinations of multiple approaches to develop the next generation of intelligent systems. A fundamental stimulus to the investigations of Hybrid Intelligent Systems (HIS) is the awareness that combined approaches will be necessary if the remaining tough problems in artificial intelligence are to be solved. Neural computing, machine learning, fuzzy logic, evolutionary algorithms, agent-based methods, among others, have been established and shown their strength and drawbacks. Recently, hybrid intelligent systems are getting popular due to their capabilities in handling several real world complexities involving imprecision, uncertainty and vagueness. [11]

The concept infers the cross pollination of any of the following disciplines

- Neuro-fuzzy systems
- Hybrid connectionist-symbolic models
- Fuzzy expert systems
- Connectionist expert systems
- Evolutionary neural networks
- Genetic fuzzy systems
- Rough fuzzy hybridization

• Reinforcement learning with fuzzy, neural, or evolutionary methods as well as symbolic reasoning methods.

> From the cognitive science perspective, every natural intelligent system is hybrid because it performs mental operations on both the symbolic and subsymbolic levels. For the past few years there have been an increasing discussions of the importance of Artificial Intelligence Systems Integration. Based on notions that there have already been created simple and specific AI systems (such as systems for computer vision, speech synthesis, etc., or software that employs some of the models mentioned above), and now is the time for integration to create broad AI systems. Proponents of this approach are researchers such as Marvin Minsky, Ron Sun, Aaron Sloman, and Michael A. Arbib. [114]

Putting the two together: Hybrid Multi-Agent Systems Hybrid-Learning Methods for Stock Index Modeling

Building on Settles and Rylander's work [7] of a PSO based neural network learning, Abraham and Chen…

> …investigated how the seemingly chaotic behavior of stock markets could be well represented using several soft computing techniques. Authors considered the flexible neural tree algorithm, a wavelet neural network, local linear wavelet neural network and finally a feed-forward artificial neural network. [3]

Their work revealed that PSO played a significant role in optimizing parameter tuning and maximizing performance, moving one step in the direction of merging the two fields of a MAS with hybrid intelligence.

In summary, the study and adaptation of nature based solutions in technical domains has proven very effective especially as the problem spaces become more complex and dynamic. Although researchers have been touting these approaches for some time now, only recently have computing resources and programming languages and frameworks allowed for effective modelling, creation, testing, and implementation of these methods. It should prove to be a very fertile and fruitful field of study.

E. Tools needed to repeat this research

The procedures to repeat our research have been captured and documented in as much detail as possible (see the Approach chapter) in an effort to open the doors to others to perform and benefit from it in their own environments and potentially build upon this work. Below is a list of recommended tools one would need (at the current time) and some notes on each one, in order to repeat or build upon this work.

1) Wireshark

One of the most readily available and free tools to use for network data capture is Wireshark, based on the libPCAP/WinPCAP application program interface (API). The Lawrence Berkeley Laboratory's Network Research Group originally developed the popular network sniffing and traffic capture tool, tcpdump, along with the libPCAP library which generates PCAPs (Packet Captures) [25]. This format has become the de-facto standard in its arena and enjoys significant community support, being used by many if not most open and closed source programs as the output of choice (i.e. Wireshark, ettercap, Snort [93], nmap [24], Microsoft Network Monitor 3.x [115]). This is due, in part, to its open API and its ability to comprehensively portray traffic traversing a network.

Unfortunately, in efforts to find any useful traces of activity which would point to digital intruders, it could take days to manually sift and sort through the large volume of data captured in a PCAP, even if capturing only a few minutes' worth on a moderately busy network. So, despite the ease with which a network administrator can capture this data, making any use of it presents an oftentimes insurmountable barrier. Tools exist to make this onerous task less burdensome but the most efficient of them are costly, well beyond the means of a small or even medium sized business.

2) ***Weka***

The open source datamining framework, Weka, was the tool we used for employing and testing the traditional classify, cluster and association algorithms. Weka was chosen for a variety of reasons, not the least of which is because it is "well suited for developing new machine learning schemes." [116] It also enjoys widespread community support and has many algorithms from which to choose and was recommended by a trusted advisor at the outset of this endeavor.

Using Weka, we established a baseline of results for traditional datamining sweeps through data, to show how (at the very least) each one would quantify the same data set. The measures of effectiveness for each algorithm are spelled out in section IV Results. Of the three types of algorithms we report results on the following implementations:

- Classifiers - J48 Tree [48] and Naïve Bayes
- Clustering – SimpleKMeans and Cobweb
- Association – Apriori

3) ***pdml2arff.py***

Out of the box, there is no way to take a PCAP from Wireshark and load it into Weka for data mining. In the process of this research, the pdml2arff.py application was written to bridge this

gap. Briefly, one exports a PCAP to a PDML formatted file using tshark, a tool provided by the makers of Wireshark and the PDML (Packet Details Markup Language) file is converted into "*.arff" which is Weka's native format. This procedure is explained in the Approach section of this paper and in the application's notes in Appendix A in greater detail. This software is freely downloadable from this url: https://github.com/abujed

4) ***Spade***

One readily available and free multi-agent platform which implements the FIPA standard is SPADE (Smart Python multi-Agent Development Environment) [21]. SPADE is a Python [117] based, FIPA compliant multi-agent framework. The overall architecture is outlined in Figure 6.

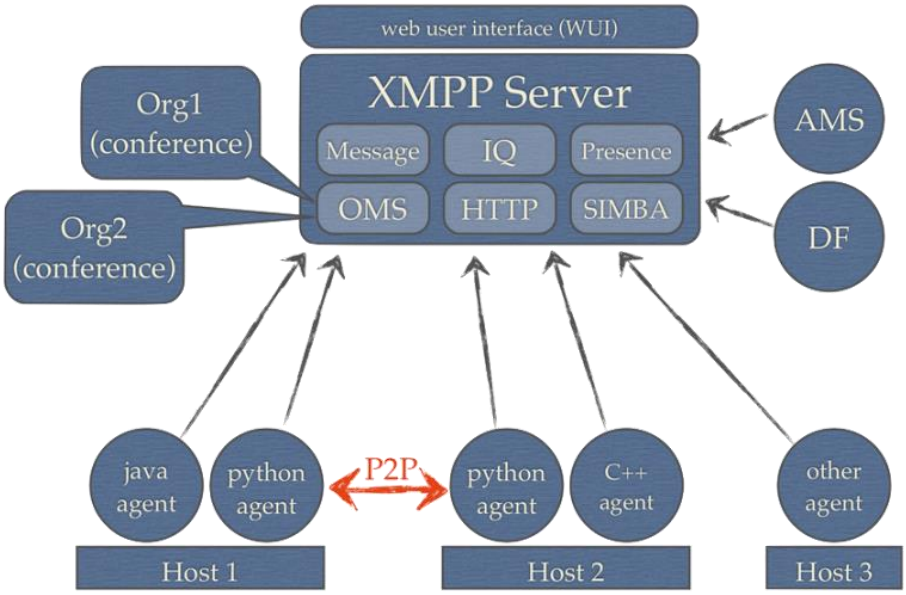

Figure 6: SPADE architecture

Spade provided the framework within which our HI/MAS was constructed. These agents can be freely downloaded from this url: https://github.com/abujed from the himasmiad repository. Spade itself can be downloaded from: http://code.google.com/p/spade2/ and installation and configuration instructions are documented in the downloads and can also be found in the Approach section of this paper.

F. Chapter Summary

In this chapter we discussed some of the history of datamining, as attempts to grapple with the "coping with information overload" problem, as well as the current state of the art and future trends in the field. Intrusion detection systems were examined, which are effectively self-contained, usually black box data miners themselves, applied to a single domain. We discussed the difficult technological gap faced by the layman network or security engineer; the ease of collecting large amounts of pertinent data, but the difficulty in gaining intimate insight into it. We examined the benefits, background and evolution, as well as various implementations of multi-agent systems, and finished with an enumeration and discussion of the tools necessary to duplicate our research.

III. APPROACH

Covered in this chapter…

- Building a lab
- Gathering and generating a suitable data set
- The path to converting PCAPs to ARFF
- Mining the data set
- Building the multi-agent systems

A. Overview of the approach

In this section we outline, in as much detail as possible, the steps required to repeat the research documented in this paper. As with all previous work [101] [102] [118] [119], one of our pre-eminent goals is to make the practical outworking of this research tractable, repeatable, applicable and malleable within the readers' own environment. To that end, the Hybrid Intelligence/Multi-Agent System we built is checked into Github.com (make sure link is good!!!) under the username "abujed", under the "himasmiad" (hybrid intelligence multi-agent system for mining information assurance data) repository where it can be freely downloaded and/or contributed to. Our pdml2arff.py application [118], can also be downloaded from there, it is in its own self referencing repository.

This research was conducted within the intrusion detection system (IDS) domain, which provided a rich environment wherein massive amounts of data could be easily generated in a controlled fashion. The data was seeded with "needles of information" as necessary to provide an excellent experimental environment. The IDS domain has several well-known and painful shortcomings as previously outlined and their measures of success or failure are easily quantified.

This provided an excellent backdrop against which our hypotheses could be tested. The research was limited to the examination of offline data.

B. Building a lab environment

A lab network (Figure 7) was constructed to mimic a real world corporate (internal) network, with a DMZ, a simulated Internet (external network) and intruders conducting cyber-attacks. This network facilitated the creation of suitable amounts of "seeded" sample intrusion detection data (PCAPs) [25] for quantifiable testing of follow on datamining systems.

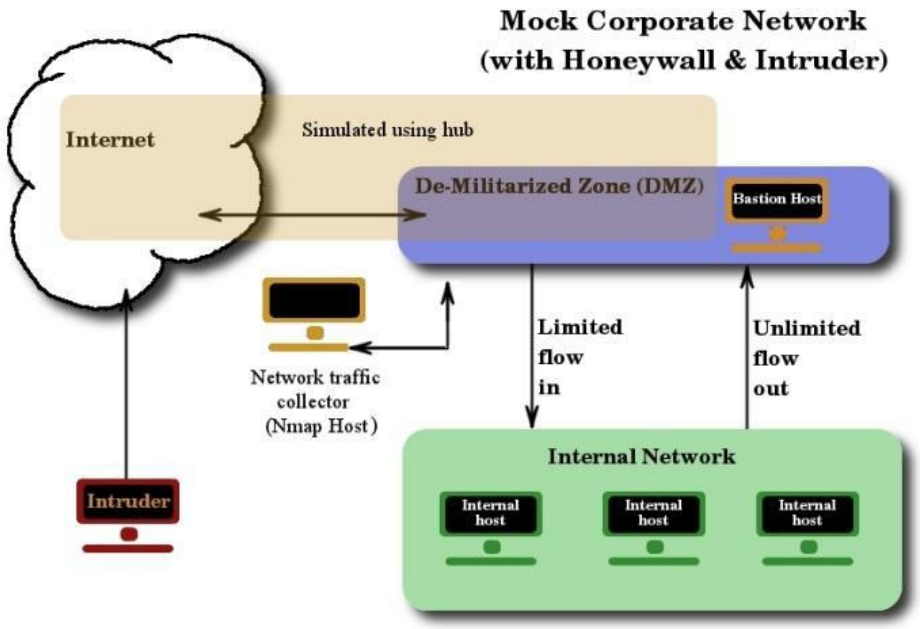

Figure 7: Lab network for generating seeded data

As part of the lab environment, both Weka and Spade were installed on an Ubuntu Linux server [23]. The instructions for the acquisition, installation, and configuration of each of these packages can be found in Appendix F - Software installation instructions.

C. Acquiring data for the research

Data useful for intrusion detection can come in many forms such as: application and system logs, IDS and IPS output, audit files and system integrity reports to name a few. These files can either be binaries or textual in format. PCAPs were used exclusively for this research for a number of reasons, which are listed below. Data for these exercises was acquired in several ways, some was generated in the lab set up for that purpose, other data was downloaded from the Internet.

1) *Why PCAPs?*

While there were a handful of reasons for solely employing PCAPs for this research, the most compelling ones are enumerated here.

Mapping the process and data to Abraham's steps 1 & 2

In regards to Abraham's [3] step 1 (understanding the application domain), the author chose to work with information assurance/network intrusion data since that is our primary area of expertise. In regards to step 2 (creating target data set), we used PCAPs as our data source. In the sections below we flesh out how they were generated and seeded. These processes were also described in previous research [101] [102] [118]. Publicly available PCAPs were also employed, some of which have been used in the research of others [48]. In this research we operated on data from both of these sources: downloaded and created in the lab.

Ease of repeatability for others

One of the chief aims of this undertaking is to provide results demonstrating the benefits of a new methodology as well as develop and document repeatable procedures which would make these benefits relevant and tractable to the everyday system and network administrator. This standardized, portable format facilitates this purpose very well.

Unadulterated Data

PCAPs contain the rawest, most complete and unfiltered representation of network traffic data available. All other data sources, such as log files from an IDS or audit files are the product of an application or device which has performed some sort of processing on the raw data. In these cases, even with the best documentation, it is unknown what data or fields might have been altered, combined, interpolated or gone missing altogether [19].

Easy to test with widely available data sets

Well-known intrusion detection data sets serve as an excellent community-wide baseline for demonstrating improvements gained through the use of a tweaked or new algorithm, but our research represents more of a paradigm shift. Instead of incrementally (or even drastically) improving upon a single algorithm, these efforts advocate a novel multi-faceted approach to improving network security, using already extant tools, giving a user access and insight into their own raw data. To be effective, the cyber-warfighter in the trenches needs fresh, accessible, on the fly insight into their own network's data, in its most unadulterated form, which this research aims to provide.

Scalability/Portability for future endeavors

Finally, while this current undertaking works only with PCAPs now, the data format (tcpcump [25]) is same in a live, streaming capture. This lays the groundwork towards real-time processing and inspection based upon the methods demonstrated.

2) ***Generating seeded research data***

In order to generate data seeded with known exploits in the lab environment, a variety of typical hacker or penetration tester actions were initiated from the "intruder," directed at the "bastion host" or an "internal host" (see Figure 7). These network activities were captured by the

"network traffic collector/Wireshark host" (e.g. Honeywall [120]) and saved as PCAPs. These files then contained known anomalies (odd flags, malformed packets, etc.). Given the maturity and excellent documentation of both Wireshark and Honeywall, it was a fairly trivial process to generate very large files (seeded or not), suitable for this type of research. Additionally, after everything was set up, it was also straightforward to seed the data with known "needles" or nuggets of information, the exploits, or hacks. This process lent itself well to establishing a control environment for experimentation and discrete measuring.

3) ***Acquisition of other research data***

In addition to custom generated data, the research also benefited from mining publicly available PCAPs which were downloaded from sites such as Evilfingers.com [121], Contagio [122], and openpacket.org [123]. These files are standardized, some are used by other researchers [124] [125] [48] and, as discussed, the format is used extensively throughout the information security community, making the test data portable and easily accessible to any collaborators or follow on work [126]. They also contain a variety of known and unknown exploits and sometimes come with additional, useful metadata. One potential drawback however, crops up when attempting to fold the packets into PCAPs from a network with a different IP addressing scheme. Depending on the type of mining to be done, extra pre-processing may need to be done in order to make the traffic appear to be in the same network. A different IP addressing scheme could stand out like a sore thumb, overshadowing the exploit which is really the intended target. In this instance, interesting patterns would be found for sure, but for the wrong reasons.

4) ***Data specific to this paper***

For the purposes of this paper we used the PCAPs listed below as our raw data sources.

From openpacket.org [123] (adapting Markey's [48] procedures):

- example.com-1.pcap (normal traffic)
- example.com-3.pcap (normal traffic)
- example.com-4.pcap (normal traffic)
- example.com-5.pcap (normal traffic)
- example.com-6.pcap (normal traffic)
- example.com-7.pcap (normal traffic)
- zeus-sample-2.pcap (malicious traffic - botnet)
- zeus-sample-3.pcap (malicious traffic - botnet)
- 12b0c78f05f33fe25e08addc60bd9b7c.pcap (malicious traffic - Kraken bot traffic)

PCAPs generated in our lab

- nmap.scan.37.pcap (vertical scan)
- nmap.scan.40.pcap (vertical scan)
- vertical.scan.raw.pcap (vertical scan)
- horizontal.scan.raw.pcap (horizontal scan)

PCAPs from evilfingers.com [121]:

- Adobe_Reader_7.0.8.0_AcroPDF.dll_Internet_Explorer_Denial_of_Service.pcap (malicious traffic)

D. Backing out of the rabbit trail – implementing a solution

As discussed earlier, Weka cannot ingest PCAPs and there was no extant utility to convert them in their entirety. The sections below describe some of the problems which needed to be overcome and how this was accomplished.

1) PCAP benefits & attendant difficulties

One factor that complicates converting PCAP files into ARFF and preprocessing them for use in Weka lies in one of the PCAP files' strengths: "Wireshark's most powerful feature is its vast array of display filters (over 141000 fields in 1000 protocols as of version 1.10.3)." [127] This number of fields continues to grow as new decodes for protocols are added. Normally any one given PCAP will not contain all 141,000+ fields, but a subset will be present and, except in the

case of the most tightly controlled of networks, it is difficult to predict what fields (and sub-fields) will appear. Given this vast array of currently available, open ended, and ultimately unknown number of displayable fields, our software was designed to read through any given PCAP and dynamically build a list of only the fields/sub-fields present in that PCAP.

2) ARFF characteristics & how to address them

ARFF is a non-normalized flat file representation of data. An unavoidable by-product of this is that all of the fields or columns must be represented on every line, potentially leaving many or most fields on a given line with an "unknown" or "missing" value. Different algorithms in Weka handle unknown values differently, so in preprocessing these must be massaged, depending on which algorithm is being used at the time (e.g. substituting hostnames for ip addresses, or vice versa, or "replacing missing values with means and modes" etc.) [3]. These functions have currently been pushed into preprocessing, after our tool performs the PDML to ARFF conversion. Future iterations of the software could add logic to handle these changes on the fly, for example, "missing values" could be given a value, which would work with the anticipated follow-on algorithm and data types could be specified at run time. To reiterate, this tool, pdml2arff.py, was written since none other was found which would convert the entire contents of a PCAP to ARFF.

3) Using pdml2arff.py

While all testing of pdml2arff.py in this research was performed on Linux based hosts, there should be no reason why it would not work in any operating system which has the prerequisite applications installed and operating properly. Assuming that one already has their interesting PCAPs on one hand, and Weka ready to ingest an ARFF on the other, the only additional applications required are tshark, Python 2.7, a copy of our application (pdml2arff.py), and

possibly a text editor. The operation of our tool is relatively simple but there are a few steps involved.

To begin, the PCAP must be converted to a PDML file. This is accomplished by issuing the following tshark command:

tshark –T pdml –r <infile> > <outfile>

where <infile> is the interesting PCAP and <outfile> is the name which will be used for the resultant PDML file.

The second step is to run the following pdml2arff.py command:

pdml2arff.py <outfile>

where <outfile> is the PDML which was created in the previous step. This generates an ARFF file named <outfile>.arff. If the file had the extension .pdml it will now be stripped off and replaced with .arff.

4) ***Loading the new file into Weka***

Assuming there are no errors in the data, this file should load without error into Weka. However, like any other ARFF file, depending on which algorithm one intends to use in Weka, further preprocessing, using any of a variety of tools, will be required. This process might include changing the default missing value placeholder ("?") to something else, converting STRING to another data type, replacing text with numerals or vice versa. The file data could also be loaded into a database, where further preprocessing can occur. In future versions, switches could be added which would allow the output to vary based on which Weka algorithm is planned for use; this would eliminate the need for the text editing in this step. In any case, after following these steps one should have the entire contents of a PCAP in ARFF format.

E. Mining the Data - Mapping the research to Abraham's knowledge discovery steps:

Using Weka [29], an open source datamining framework, we closely followed Abraham's knowledge discovery steps, generating results from three datamining algorithm types: classifying, clustering and associating.

While simple to write out, most every step of the datamining process is quite involved and exceedingly critical. Each one builds on the one preceding it and careful attention must be paid to each one to ensure that the follow on processes will be accurate and yield the desired results (i.e. the nuggets of knowledge that we ultimately seek). Successful performance of these steps requires domain knowledge (in our case, information security/intrusion detection), and the collection of meaningful data, which is "clean" and can be well understood by the algorithms for training, or has attributes suitable for meaningful interpretation. The mapping of our processes to Abraham's [3] steps is elaborated upon below:

1) ***Step 1. "Understanding of the application domain…"***

The Literature Review chapter outlined the development and understanding of the application domain, and the relevant prior knowledge. For this research we identify the goal of knowledge discovery as finding the intruders, network traffic anomalies or malicious activity in the seeded data.

2) ***Step*** 2. "Creating target data set..."

In this step, we established a set of PCAPs which were either generated and captured, or downloaded (as described in section C of this chapter) to serve as training and test sets to be used in Weka. Sources such as openpacket.org [123] and evilfingers.com [121] provided suitable data which had also been used in the previous research of others which we leveraged for our endeavor [48]. Each algorithm had this common set of PCAPs (listed in section C.4 of this chapter) run

through them to see what each one would (or would not) find. From this controlled data, concrete metrics could be drawn to enumerate the strengths and weaknesses of various types of algorithms.

Jumping ahead a bit in the analysis of the results, in leveraging the research of others, we did not expect to find anything more than they did when simply repeating their experiments. However, in the larger context of this research, we certainly expected to be able to tease more knowledge out of a set of PCAPs when applying a given researcher's methodology as just one of several in a set of tools. We believe the results and their analysis, discussed below, support these expectations. Furthermore, our HI/MAS framework, to which future tools can be readily added, provides a novel and dynamic system offering a powerful multi-dimensional set of operators to combat the increasing complexity of the coping with information overload problem.

3) ***Step*** 3. "Data cleaning and preprocessing…"

Abraham's Steps 3 (preprocessing) and 4 (data reduction and projection) are widely known to be the most labor intensive [20]. Preprocessing the data varies from algorithm to algorithm and we address the specific procedures and methods taken for each one in turn below.

This step consisted of multiple, time consuming operations such as removing noise, handling missing data fields and ensuring the validity of the data. In addition to custom scripts written by the author, some of the tools used for this step are AtoZ, GTVS and fullstats [30]. Additional tasks in this step usually also include reformatting of data from its original format into the various formats expected by the datamining programs. In our case this meant either extracting the relevant fields from a PCAP, to csv, to ARFF format for Weka [29] or using our pdml2arff.py tool with the same resultant Weka input file. The importance of the pre-data-

mining (preprocessing) phase and its sub-steps cannot be underestimated or done in a haphazard manner.

4) ***Step 4a. "Finding useful features to represent the data"***

At this stage of the process one of several things happened, either extraneous data was pared out or attributes (such as unique keys) or additional information not included in the collected data but known to the researcher (typically metadata) was added in.

This last step is used as an aid for data reduction and projection, as it provides a first shot and quick look at the data after it has been mined, validating (or invalidating) that the chosen features will be suitable for mining by machine intelligence.

5) ***Step 5. "Matching the goals … (The hinge of our thesis)***

Step 5, or "matching the goals … to a particular datamining method" is the hinge upon which our thesis turns. We posit that, examining a given data set, while tweaking the goals slightly, according to the strength of each type of algorithm, will yield better overall insight into the problem space than that provided by one algorithm alone. We believe the results of our tests (section IV) show this to be true.

The literature in the "datamining for network intrusion" field is replete with new and innovative types of improved datamining algorithms and we look forward to employing them in future work. For this round of research, in our efforts to make the procedures accessible and repeatable across a wide spectrum of users, it was decided to use readily available, open source packages, requiring little or no customization or fiddling with to get them working off the shelf.

In order to test our hypothesis that 'more algorithm variety yields more knowledge', we took a common set of data (PCAPs) and ran them all through Weka using the following algorithms:

J48 Tree and NaiveBayes (classifiers), CobWeb and Simple Kmeans (clusterers), and finally, Apriori (associator).

F. Walking through Abraham's steps with our own data

The following sections describe in detail the steps undertaken in running our data through each of the three different datamining lanes: classifying, clustering and association.

1) *Classifying*

Preprocessing

While there are a number of ways to accomplish the tasks below, for the classification exercise we closely followed the steps for preprocessing and datamining enumerated by Markey and Atlasis [48]. Using tcptrace, all of the interesting fields were extracted from the PCAPs yielding csv files containing *n* attributes. From this csv file, all but 7 attributes were removed which were: port_a, port_b, total_packets_a2b, total_packets_b2a, unique_bytes_sent_a2b, unique_bytes_sent_b2a and session_duration. An 8th attribute was added labeled "Class." The "Class" attribute was of data type "nominal" and assigned a value of either "NORMAL", "MALICIOUS" or "SCANNING", depending on which PCAP the data was derived from. We initially crafted a configurable script (markeypcapmod – found in Appendix A) to accomplish the repetitive tasks outlined in Markey's paper. The script needed to be run on each type of PCAP (e.g. malicious, normal, scanning), the results of which were then to be combined as directed in the script's internal documentation. These scripts were later coded into the agents in the multi-agent system.

Datamining

After the script extracted the interesting fields from the PCAPs and converted them into csv format, the resulting files were loaded into Weka through the explorer interface. They were then

saved in ARFF format and Markey's directions were followed in running the J48 Tree classifier. The classifier results were then saved as a model with which to later classify unknown or unlabeled data. One of the base set of PCAPs seeded with an exploit was then run against that model in order to test its efficacy.

The classifiers were also trained and tested with PCAPs from the test set which had been seeded with nmap scans (*vertical* and *horizontal*).

The complete results from the classification run can be seen in in detail in Appendix B - Run Results and are discussed in section IV Results.

2) Clustering

For reasons discussed in the Literature Review Clustering section, coupled with the shortfalls of the classifiers to identify scanning traffic, (demonstrated in the Results: Classifiers section from classifier experiments), clustering was used to elucidate patterns of communication between hosts (e.g. scanning). These algorithms, which model data descriptively, work differently from classification algorithms (predictors), so although the same set of PCAPs were used, running data which had been preprocessed for classification through a clustering algorithm would have yielded results which would not be very useful. With this in mind and with a familiarity of the nature of PCAP data, as well as the strengths of the clustering algorithms, the data was pre-processed in a manner which made effective use of the strengths of the chosen clustering algorithm. The practical outworking of this meant that a different set of pre-processing steps (outlined below) were required to render the data into a suitable format to yield worthwhile results.

The outcomes of these efforts are discussed in the clustering section in the Results section. The steps below outline how the data was preprocessed and then run through the clustering algorithms (CobWeb and SimpleKMeans) in Weka.

Specific application of clustering to our research

In order to best enumerate the one to one (vertical), or one to many (horizontal) communicant relationships, or rather, to see who's talking to whom on a network, it was best to filter out the noise in the PCAP files and retain only the columns or attributes useful to the clustering algorithms. For purposes of this research we kept only the following columns: ip.src, ip.dst. The src/dst pairs provided data for the clustering algorithm to group and Weka internally provides its own way to uniquely identify each record, obviating the need for keeping any columns like packet_id. To be sure, there are many other ways to slice and dice the data to find interesting clusters, this is but one, and the results (Results:Clustering) proved very insightful.

Preprocessing

As with the classifier exercise, the initial step for clustering consisted of file conversion, however in this case a different procedure was used, which did not employ tcptrace. The command line arguments spelled out in Appendix A in the usage notes for our pdml2arff.py application [118] were used to convert the interesting PCAP to PDML, then the resultant PDML was converted to .arff with pdml2arff.py. Depending on the size of the PCAP and the computing resources available, these steps could take a while. In the resulting ARFF, all of the PCAP's attributes will be output with the datatype of "String" which may or may not need to be changed, depending on the follow-on algorithm being used. That value *did* need to be changed for this clustering exercise.

Column Selection

To strip out the unwanted columns and leave only the interesting ones (outlined above), the ARFF file was opened in Weka, and the two interesting columns were selected: ip.src, ip.dst, as shown in Figure 8.

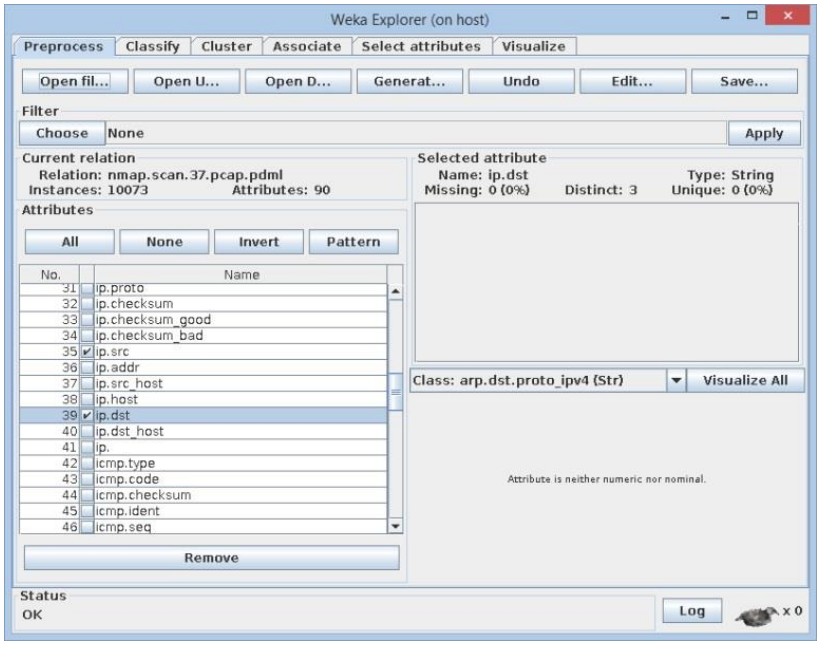

Figure 8: Column selection (a)

Selecting the "Invert" button, highlights every other attribute EXCEPT the two we wish to keep. Selecting the "Remove" button removes all of the undesired attributes or columns. (Figure 9)

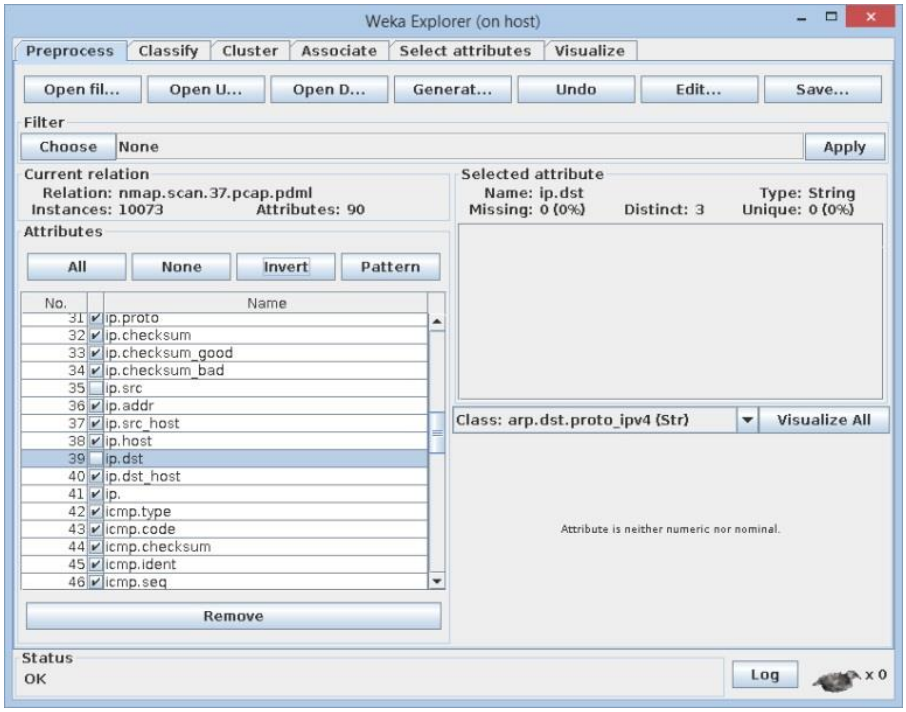

Figure 9: Column selection (b)

Column Reduction and Cleanup

The only remaining columns should be the two previously selected as shown in Figure 10.

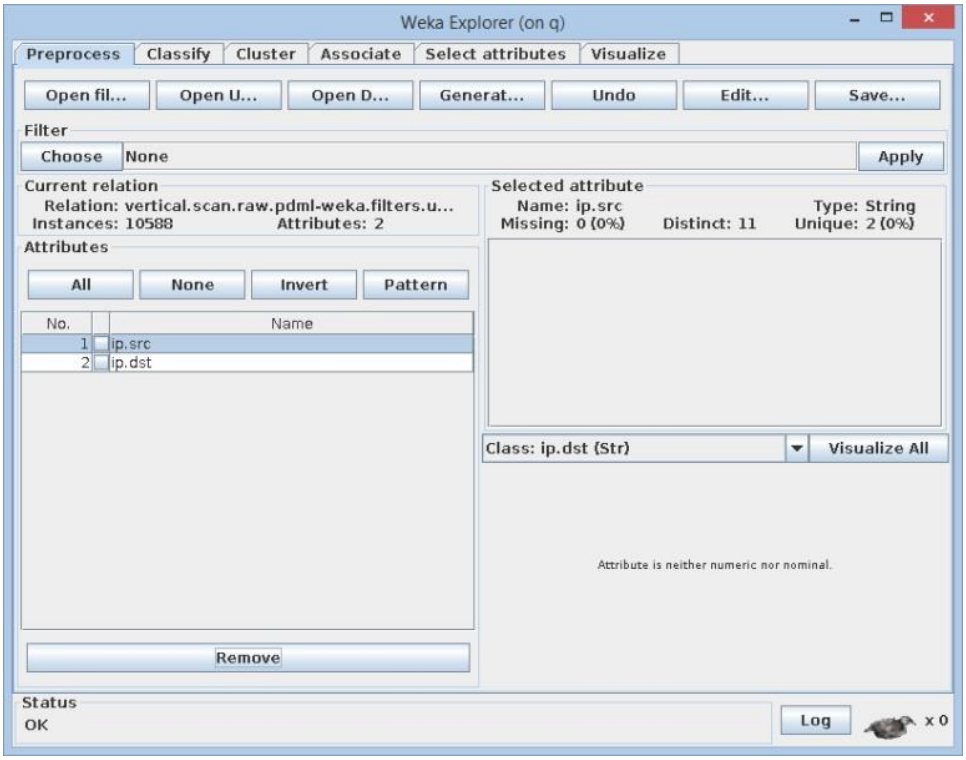

Figure 10: Column reduction

At this point Weka has renamed the Relation which is being stored in memory. It should be saved as a new ARFF file by clicking the "Save" button in the upper right hand corner and should be given a suitable name.

Opening the new ARFF file in a text editor should reveal something very similar to Figure 11, which shows the multiple instances of data with unknown or missing values.

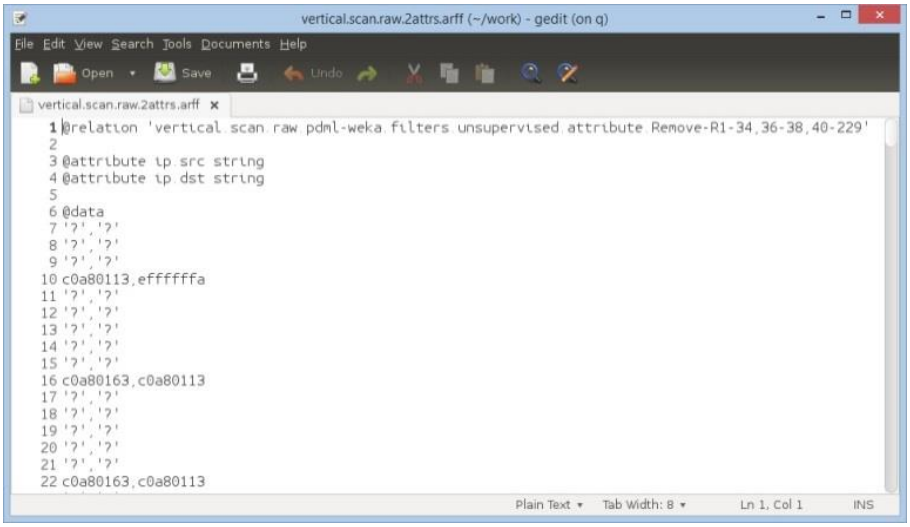

Figure 11: ARFF for clustering before merging down packet id

If instances remain which have only missing values, represented by lines which contain only '?' or "?", these lines should be removed. At this point, if so desired, the @relation can be renamed and the file should look similar to Figure 12.

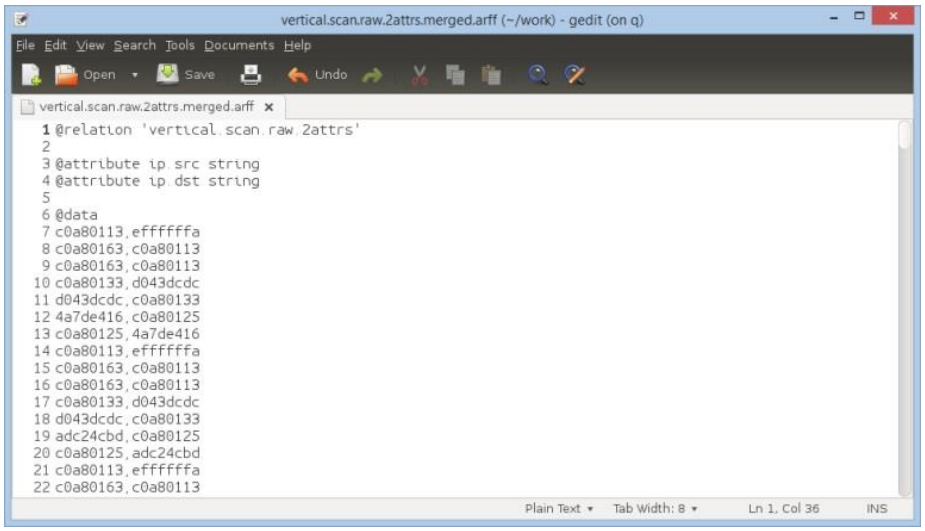

Figure 12: ARFF for clustering after removing missing values

Attribute data type modification

Some algorithms, like SimpleKMeans, cannot operate on certain datatypes, like "string" which is the default output of pdml2arff.py. Fortunately, Weka's built-in preprocessing toolset

makes converting the datatype to something useful fairly simple, if not straightforward. In order to convert "string" to "nominal" the data must be loaded in Weka Explorer, then filters->unsupervised->attribute-> StringToNominal should be chosen. Clicking in the field where "StringToNominal" appears and an object editor will open up (Figure 13). From here, the range of attributes one wishes to have the filter operate on can be selected. In our example, both of the attributes need to be converted from String to Nominal so we set the range to encompass the appropriate attributes as shown in Figure 13 (nb for this operation Weka counts from 1 – not 0).

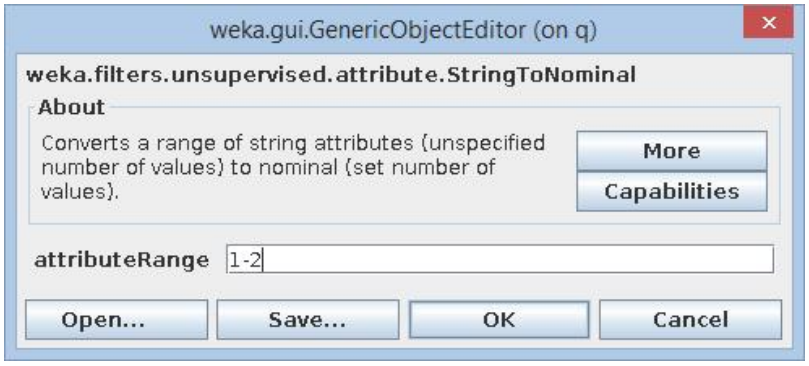

Figure 13: String to nominal requestor

Following this setting adjustment, clicking OK will close the GenericObjectEditor requestor. Clicking "Apply" will result in the selected range of attributes (previously of "String" datatype) to be converted to the new datatype "Nominal."

In some cases it is desirable to retain an attribute like "packet_id" which is known to be numeric, a data type which is acceptable to SimpleKMeans. However, Weka has no filter to convert "String" to "Numeric," so this datatype "conversion" must be performed manually. To do this, the ARFF would be saved through Weka Explorer then opened in a text editor and the data type would be changed from "string" to "numeric", then the file would be saved over itself or as a new file.

Datamining

Having completed the pre-processing as outlined above, the SimpleKMeans and Cobweb algorithms can now operate on the data. The newly saved, or just altered ARFF should be opened in Weka Explorer through the Preprocess tab. After opening the file, the "Cluster" tab should be clicked, and then one can select "Choose" on the following requestor. At this point, either "SimpleKMeans" or "Cobweb" can be selected for data mining. For purposes of this exercise, we used the same PCAPs used in the classification exercise which were seeded with vertical and horizontal nmap scans from one host to another, so we initially accepted the default setting of 2 clusters. The number of clusters to seed SimpleKMeans can be adjusted by clicking in the field which contains the name of the algorithm. This brings up a requestor where, among other things, the numClusters can be changed.

After executing the runs as described above, a number of different data arrangements and settings were tried, to include: the adding or removing of the packet_id column as well as seeding with different numbers of clusters. Additionally some runs were performed after combining the seeded data with some "normal" traffic using one of the "example.com" PCAPs used in the classifier exercise.

The results from the clustering runs can be seen in Appendix B - Run Results and are discussed in the Clustering area of the Results section.

3) ***Association***

As with the classification and clustering examples, PCAPs from the same set were used for these association exercises, in which the Apriori associating algorithm was applied.

Preprocessing directions

As with the other algorithms, the first step is to convert each PCAP into an ARFF, which is then loaded up in Weka Explorer. More attributes are kept for this algorithm than for the previous ones, but it is important to prune any fields which have duplicate or redundant data. As examples of redundant data, the pairs of attributes "ip.addr" and "ip.host", as well as ip.dst" and "ip.dst_host" contain duplicate data, as do "eth.dst.eth.ig" and "eth.dst.eth.lg" shown in Figure 14. This is the case for a number of fields/attributes and given that PCAPs are capable of displaying over 174,000 fields (as of version 1.12.0) [127], the task of paring these down can be quite laborious and time consuming.

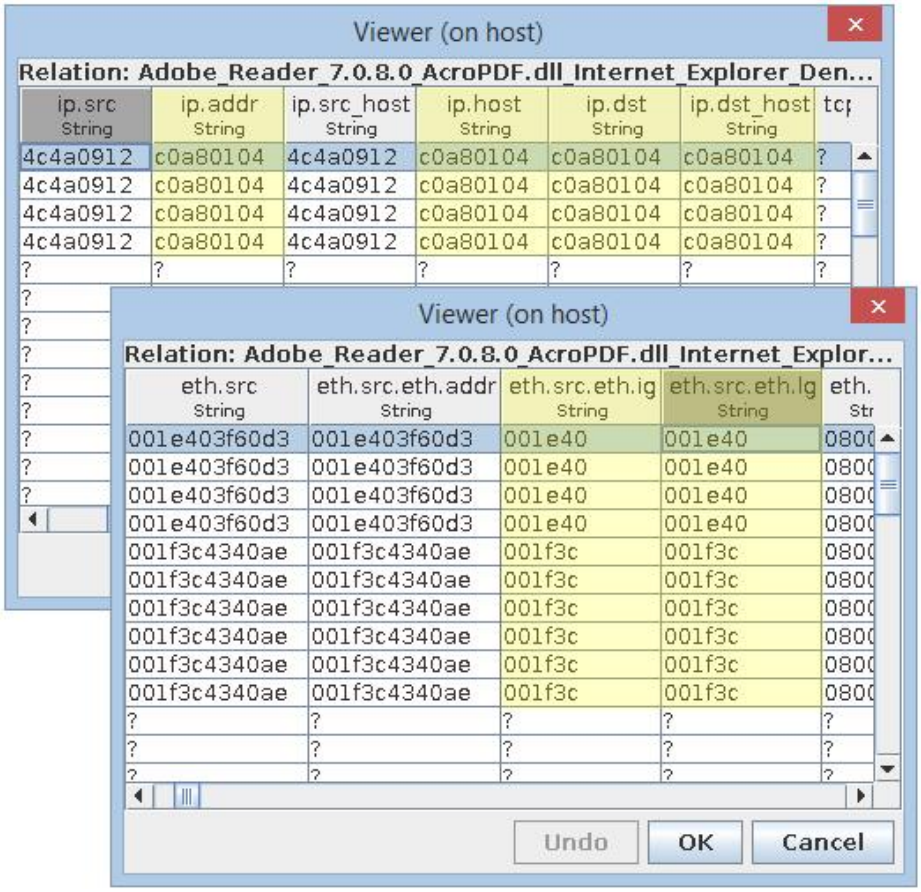

Figure 14: Identifying & pruning attributes with duplicate data

Fortunately, Weka provides a way to inspect and massage the data by clicking on the "Edit" button, which opens a data "Viewer" window (Figure 14). Right clicking on any column heading and selecting "optimal column width (all)" causes all columns to display the data to its fullest extent. In this fashion, duplicate columns can be more easily identified and can also be deleted by right clicking and selecting "Delete Attribute." Alternatively, the columns with duplicate data can be annotated and removed either as described above (Figures 8 – 10) through the main Weka Explorer window or by using the command line to delete them from the ARFF:

java weka.filters.unsupervised.attribute.Remove -R <attributes,to-remove> -i <input.arff> -o <output.arff>

Examples provided by Weka for how to define attributes to remove (placed after the –R) look like this: "first-3,5,6-10,last." The attribute numbers are listed down the left hand side in the Attributes section in the Weka Explorer window and begin counting at 1, rather than 0.

For purposes of this exercise, following in the footsteps of previous work [73], feature selection for processing began by pruning all but the following attributes (renamed to fit our naming conventions): ip.src, ip.dst, tcp.srcport, tcp.dstport, tcp.flags, tcp.flags.ack, tcp.flags.push, tcp.flags.reset, tcp.flags.syn, tcp.flags.fin. In addition, these other attributes were also included: udp.srcport, udp.dstport, udp.checksum, eth.dst.eth.addr, eth.src.eth.addr, tcp.checksum, in order to help better characterize the data.

After removing the unwanted attributes, the ARFF in memory was saved, and then the preprocess.do.merge.sh script (Appendix A) was run on it to compress all instances of a given packet_id into one line. Next, all of the attributes' datatypes were converted from string into nominal since Apriori only works with nominal data. By default, our tool pdml2arff assigns all attributes the datatype "string" which makes this step fairly straightforward. This can be done

through the Weka explorer by opening the ARFF, then clicking “Choose” under “Filter” and selecting filters -> unsupervised -> attribute -> StringToNominal. The filter will show up in the text field next to the word “Choose” with “last” designated as the default attribute to change. Left clicking in the text field (now showing “StringToNominal –R last”) brings up the filter requestor as shown in Figure 15.

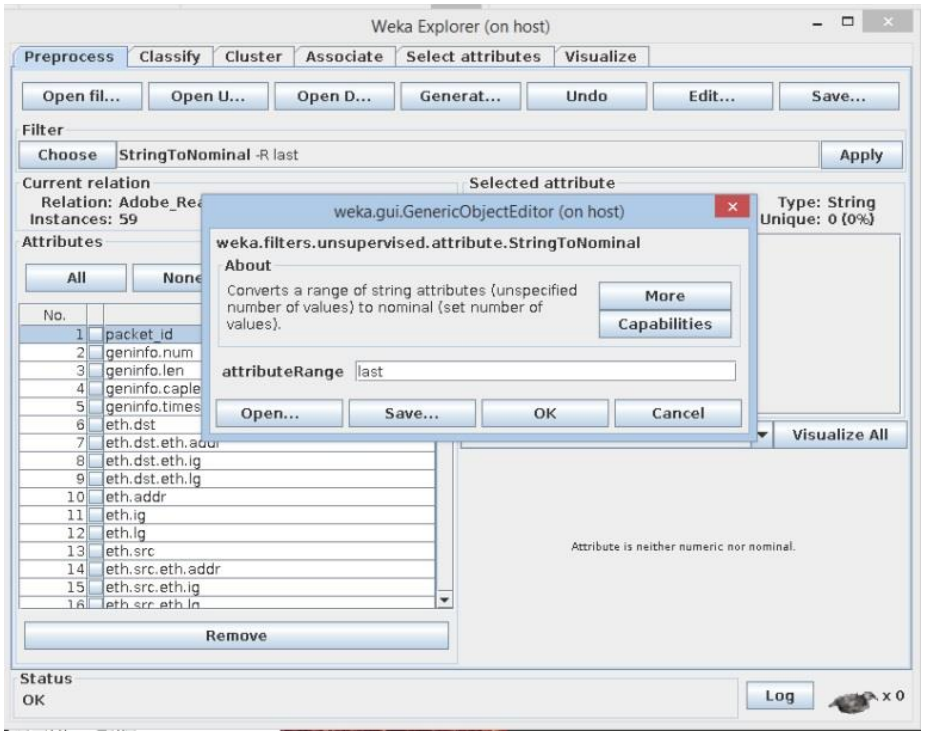

Figure 15: Converting string to nominal

Since ALL of the attributes need to be converted, replace or modify the value “last” in the “attributeRange” field with “first-last” instead, as shown in Figure 16. Then click “OK” on the requestor and this change should be reflected in the text field next to the “Choose” button. Clicking “Apply,” off to the right will run the filter, converting all of the attributes to datatype “nominal.”

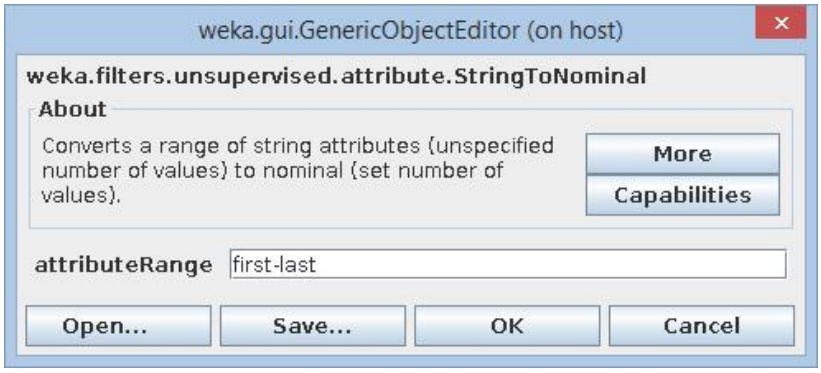

Figure 16: Modifying attributRange value

As with every operation in Weka this process can also be accomplished from the command line by issuing the following command:

java weka.filters.unsupervised.attribute.StringToNominal -R first-last -i <input.arff> -o <output.arff>

At this point the data has been successfully prepared for submission to the Apriori algorithm and the ARFF in memory can be saved with its newly nominalized attributes.

Datamining

After preprocessing, generating the mining rules themselves is fairly straightforward, although the process may take quite some time and resources. In the Weka Explorer, click on the Associate tab, then the "Choose" button, then under the "associations" folder select Apriori (Figure 17).

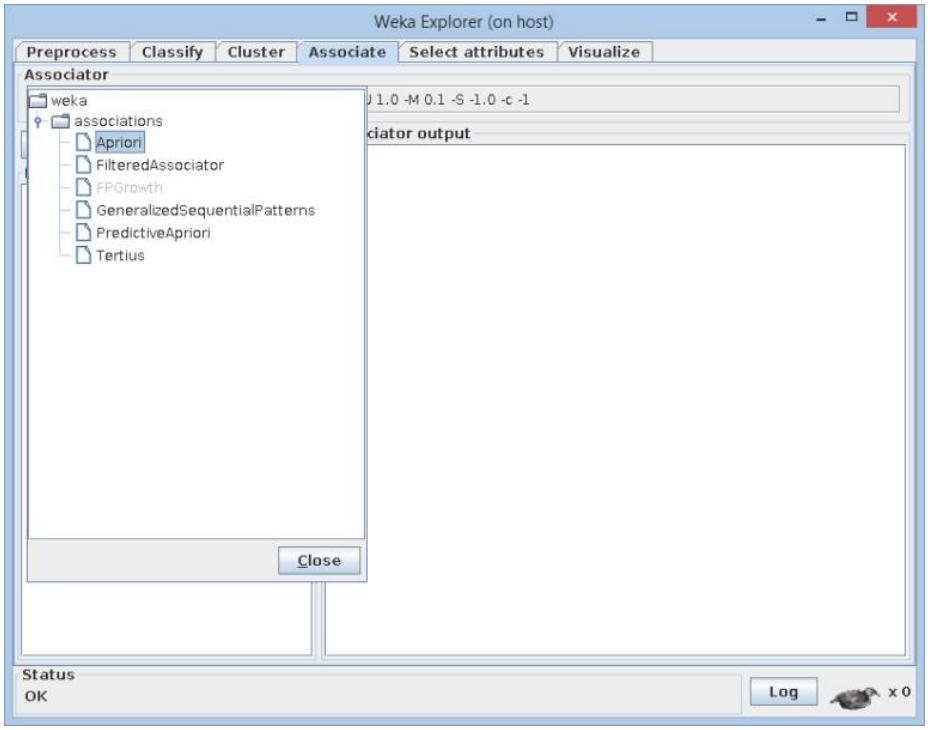

Figure 17: Selecting Apriori

The Apriori defaults should be accepted for the time being, and the "Start" button should be clicked to launch the process. Depending on a number of factors including: the computing resources and amount of computer memory, the size of the data set as well as how many attributes it has, this process could take hours or days to finish. If the bird (weka) in the lower right hand corner stops moving, something has gone wrong and you will need to either try again or tweak the data file (make it smaller), or adjust the Apriori settings.

The results from the association runs can be seen in Appendix B - Run Results and aare discussed in Results: Associator section.

G. **Creating the HI/MAS**

DeLoach, Oyenan, and Matson describe multi-agent systems (MAS) as "groups of self-directed agents working together to achieve a common goal." [12] Scientists and researchers have increasingly turned to nature for inspiration (e.g. ant and bee colonies) to provide answers

[3] [4] [5] [6] [7] as the complexity of our technological problems grows, and historical solution approaches fall short. Our intent was to demonstrate the merits of travelling the same path in the realm of intrusion detection.

After developing procedures for each of the datamining lanes and then establishing baseline results from the various implementations of conventional datamining techniques (above), a very basic HI/MAS framework was built to test the thesis that a HI/MAS could enhance the process by adding the following:

1. Effective automation of data mining tasks

2. Presentation of results for examination by a human or another agent

3. The potential for opening a future window onto stigmergic behavior of the agents, which could possibly provide unlooked for insights (as described in the Literature Review).

4. The laying of the foundation for a future intrusion detection product

To the authors' knowledge, a HI/MAS system for intrusion detection, which incorporates two or more of the various approaches (classifying, clustering, association) in order to cull nuggets of information from intrusion detection data set did not exist prior to this work. Nor could an intelligent overarching layer ("intelligence layer" in Figure 18) be found which manages, optimizes or implements human feedback into the loop. The various current techniques require knowledge seekers to frame their questions in a way that is well suited to the method at hand, changing techniques depending on the type of data or mining tool. After establishing baseline results from the various implementations of conventional datamining techniques (above), a HI/MAS framework was created to test the thesis that a HI/MAS based intrusion detection system (represented by the "intelligence layer" in Figure 18) would provide insights into the data not yielded by a traditional, single algorithm based system.

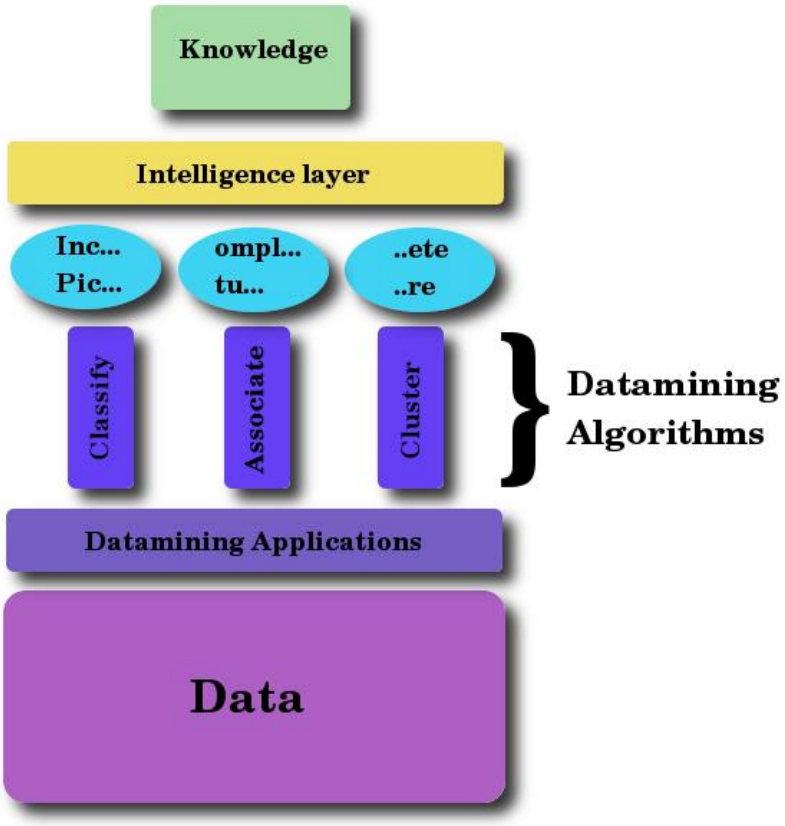

Figure 18: Traditional Datamining pillars with HI/MAS (intelligence) layer

All of the following work was performed on a server running 64bit Ubuntu Linux 14.04.1 [23] and Python version 2.7.6. On a clean installation, the following packages and their dependencies also need to be installed: Weka [29] and SPADE (as described in the Literature Review).

1) ***Building Agents***

In the design and architecting of a multi-agent system, thought must be given to the decomposition and distribution of tasks [108]. Much like object oriented programming a delicate balance must be struck in an agent's logic; it must be granular enough to perform necessary tasks, but not overfit in such a fashion as to be unable to adapt to unforeseen situations [108] [128].

Mapping tasks of this research to types of agent behavior:

- *Cyclic/Periodic*: This behavior is suitable for monitoring directories for the arrival of new PCAPs to process.

- *One-shot and time-out*: One shot behavior was implemented in our framework for both preprocessing and datamining, as shown in Figure 19. Agent classes examples. It worked very well for an agent which had one job to do.

- *Finite state machine (FSM)*: Consisting of states, inputs and outputs [129], FSMs may have a role to play in tracking previously unseen patterns as each algorithm looks at the same data in a different manner. This aspect ties into the visualization of stigmergy which we hope to realize in future research.

- *Event response*: This behavior which complements agent interactions [130], is also codified into our agents; for instance, when an agent finds a PCAP in a directory it is monitoring (event) and it notifies other agents of this discovery (response). This activity occurs in several of our agents, as they notify the next agent in the datamining lane that they have completed their task and hand off the onus to the next agent in line.

Many of the steps, scripts and snippets of code used in the creation of these agents were documented and ported from previous work [119]. Shell scripts were "Pythonized" where possible or simply wrapped in Python in order to be folded into the SPADE agents.

2) ***Data Mining with agents***

As mentioned in the Literature Review "Tools needed to repeat this research" section, Weka was used as our datamining framework. This application can be just as easily run from the command line as through its GUI. This feature lends itself well to the porting of earlier work into the domain of a multi-agent system.

Figure 19 represents an abstract view of two of the classes of agents, and how they are instantiated in the system.

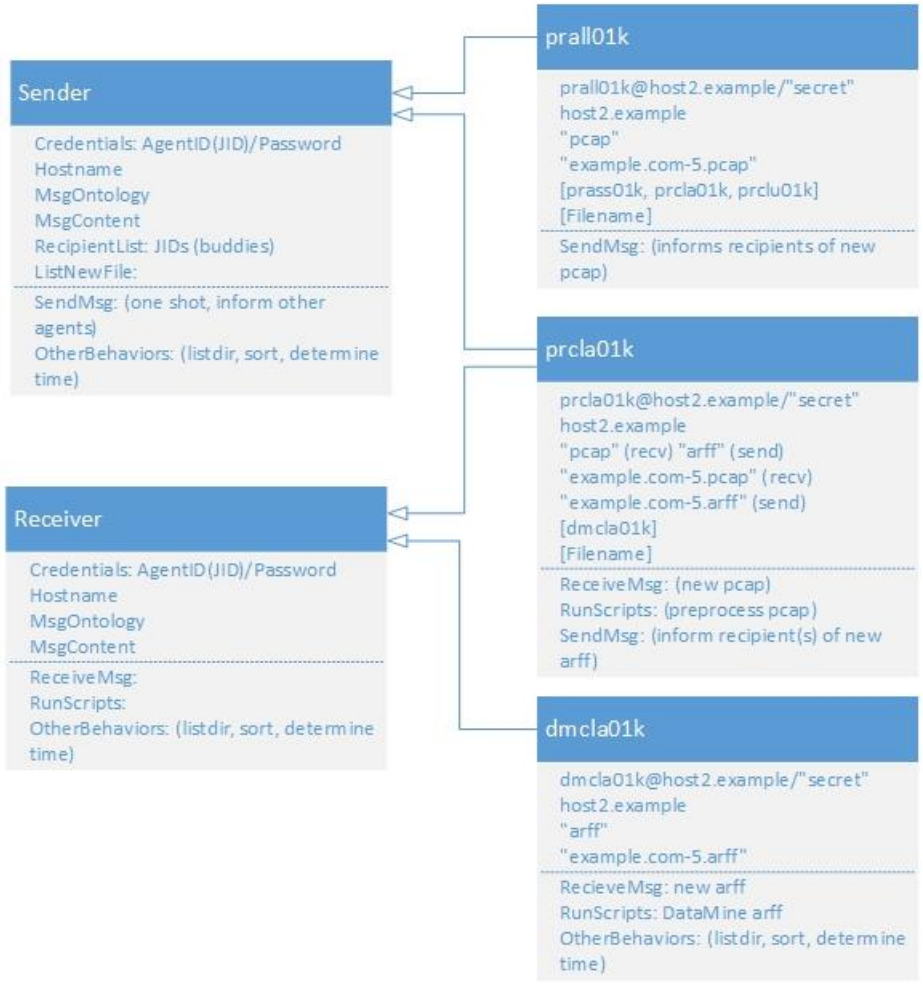

Figure 19. Agent classes examples

3) *HI/MAS data flow*

It should be noted that the steps enumerated above and below are only a small sampling of the myriad ways in which the PCAPs could be preprocessed and datamined. One of the many benefits of this HI/MAS approach is the ease with which new lanes of processing can be added to accommodate either new algorithms or different ways of slicing and dicing (preprocessing) the data for algorithms already in use.

The bird's eye view of the system's data flow is as follows: A new PCAP is dropped into a directory, where a "directory manager" agent facilitates its disbursal down (currently) 3

datamining queues, one each for classification, clustering and association. Each queue preprocesses and then mines the file in accordance with the methodologies explored and documented in previous research [119] and spelled out in detail earlier in this paper. The specifics of each lane are described below.

Additionally, although not spelled out below, as the agents walk through their procedures, the original PCAP is moved to an archive directory, and the agents rename, timestamp, and archive the files according to the stage they are currently in. This provides some self-documentation of the process through the names of the files themselves, and introduces robustness and troubleshooting aids to the system by providing a breadcrumb trail of sorts.

To reiterate the larger picture, a given PCAP goes into the HI/MAS and then percolates through the various datamining algorithms. According to each algorithm's strengths, each one outputs what it has found in a readily accessible place for review and interpretation. This hybridization of machine learning techniques provides multi-dimensional and novel insights into PCAP data, as demonstrated in the Results and Analysis and Conclusions sections.

Classifying lane

At the top of the classifying lane an agent in the preprocessing area awaits notification of the arrival of a new PCAP from the "directory manager" agent mentioned above. Upon notification of the availability of a new PCAP, the agent copies it into its local directory and preprocesses it, preparing the file for handover to the datamining agent or agents. The preprocessing steps are an automation of the steps outlined above which were initially performed manually, and are an adaptation of Markey's procedures [48].

When the preprocessing agent is finished, it hands off its file to the datamining agent or agents. In its current state our HI/MAS has two classification datamining agents, one each to repeat the

J48 and NaiveBayes datamining procedures which were performed manually and described above. Also, in its current state, the preprocessing requirements are the same for both of these datamining procedures, so each of the datamining agents listens for a notification of a new file from the one preprocessing agent.

A slight variance exists in the HI/MAS from the manual steps outlined above for the classifier in that the system is performing as if it were in a real time environment, essentially looking for misuse/signature/rule-based attacks which have already been defined. As such, the steps to train the classifier have been dropped, instead it is only checking for "known" attacks. In other words, this track would never expect to receive training data, but only test data. In this fashion this lane performs similarly to any other "signature-based" intrusion detection system.

Having said that though, given the flexibility and modular nature of this HI/MAS, there is no reason that a "training lane" could not be added. The extant agents can readily be re-tooled to process training data and create models, and a new datamining agent could be added to run test data against the new model/signature.

When finished mining, each agent places its output in a directory for review. Examples of this output are shown in the Results chapter.

Clustering lane

This lane functions almost identically to the classifier lane. A preprocessing agent waits at the head of the clustering lane to receive notification of the arrival of a new PCAP from an instance of the "directory manager agent" ahead of it. Upon receipt of this notification, the agent copies the new PCAP into its local, working directory and it preprocesses the file and then hands it off to two datamining agents for clustering. In the current HI/MAS instance the two datamining agents are using the SimpleKMeans and Cobweb algorithms, again, repeating the steps outlined above in

an automated fashion. Like the classifying agents, the two datamining agents share the output of a single preprocessing agent since they are currently programmed to ingest data in the same format. Having said that, the system could be easily configured for the opposite to be true, having several preprocessing agents, each of which preprocess the data in their own way and then hand off the files to a single datamining agent.

These agents place their output in a directory for review. Examples of their output are posted in the Results chapter.

Associating lane

The process for this lane is virtually the same as the other two. An agent at the head of the associator lane waits to receive a message from an instance of the "directory manager agent" that there is a new PCAP to be picked up. Upon receipt of the notification of a new PCAP, it copies the file into its lane, and then runs the first of two preprocessing scripts which converts the file from PCAP to PDML and then from PDML to ARFF using our pdml2arff.py tool. The second script removes some columns, then converts data in string format to nominal. After this agent is finished with the file it then hands it off to a datamining agent for associating. In the current version of our HI/MAS, only the Apriori algorithm is being used.

As with the other two lanes, it places its output in a directory for review, an example of which will be shown in the Results section.

H. Chapter Summary

In this chapter our procedures were copiously documented step by step, offering a viable process through which reasonably capable network or system administrators can examine their own data. We outlined the components necessary for building a lab suitable for generating intrusion detection data, explained how to generate and seed the data, how to convert it into

Weka's format, and walked through the steps necessary to mine the data. Finally, the process for emplacing these solutions in a multi-agent system was outlined.

Information regarding number of runs per algorithm, which data was used in a given run and the output in general from the procedures outlined above is found in the Results chapter.

IV. RESULTS

Covered in this chapter…

- PCAP extraction and conversion issues spelled out
- Multiple datamining lanes results
- HI/MAS results

In this section we post the results garnered from the procedures run in the Approach chapter with some immediate feedback and commentary; further macro level discussion and analysis can be found in the following chapter – Analysis and Conclusions. Before jumping into the datamining results, we take a look at the findings surrounding the question of file conversion between PCAP and ARFF.

A. A demonstration of the inadequacy of extant tools for PCAP to ARFF conversion.

Below is a discussion of the results of the series of tests aimed at finding out how to best extract the entire contents of a PCAP in a fashion suitable for importing into Weka. The following clearly demonstrates how much data would remain locked away using only the embedded utilities or any other extant tools. Hopefully this tool will serve others well as they seek greater insight into the nature of the traffic traversing their networks.

1) Data rich PCAPs

PCAPs contain all the information available about every packet which traverses the interface on which it was captured. It is important to remember however, that PCAPs are binary files, housing relational data which does not readily lend itself to flat file output. While Wireshark's data capture capabilities are stellar, its PCAP to .csv export features in the graphical user interface (GUI), run with default settings, leave out quite a bit of data. Figure 20 shows

Wireshark's GUI view of the PCAP:

Adobe_Reader_7.0.8.0_AcroPDF.dll_Internet_Explorer_Denial_of_ Service.PCAP downloaded from Evilfingers.com [121].

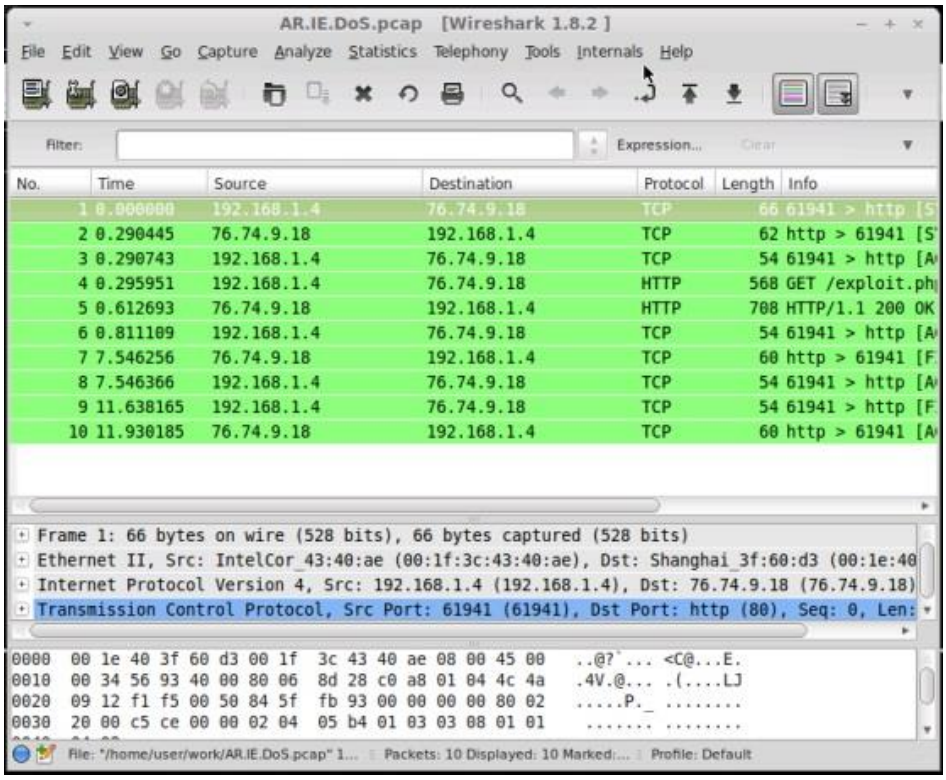

Figure 20: Wireshark showing 10 packets in the PCAP

As can be seen, there are only 10 packets in this particular PCAP. Figure 21 shows the .csv exported by Wireshark and, given the picture above, it may appear to have given us all of the pertinent data.

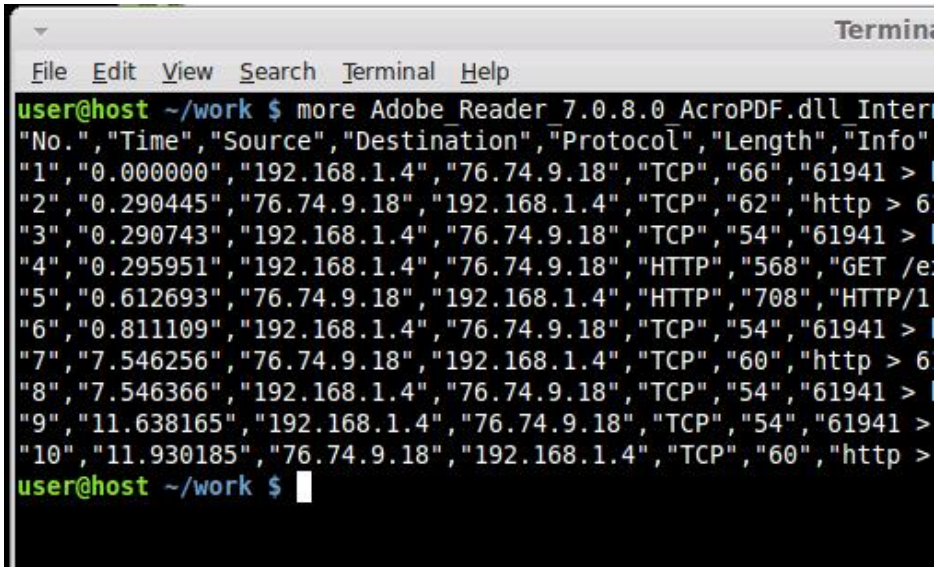

Figure 21: Output from Wireshark export to .csv (truncated)

Figure 22 shows a portion of the output from a tshark rendered PDML (xml based) output from the same PCAP.

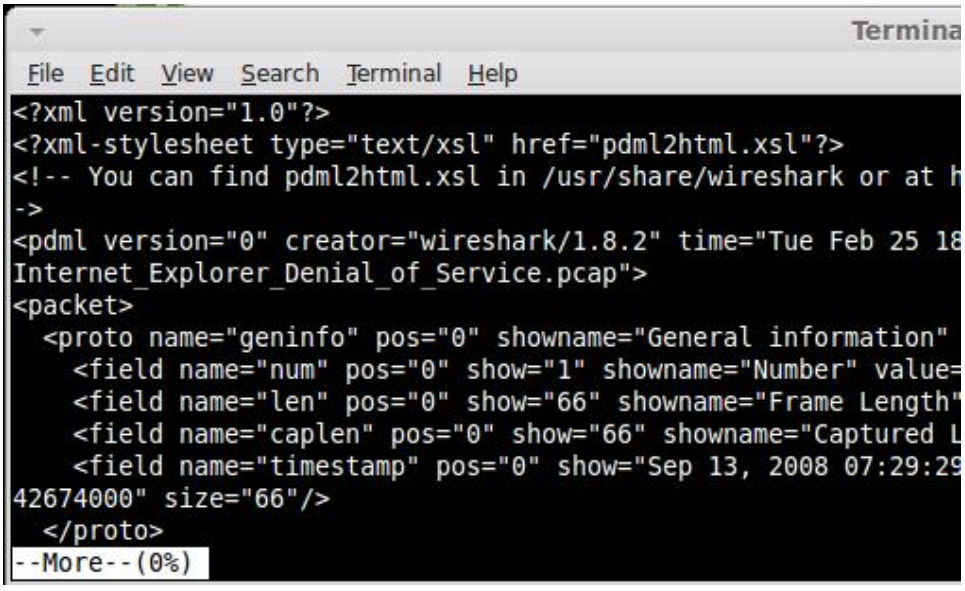

Figure 22: A portion of the PDML output (truncated)

2) *Comparing PDML to csv output*

As can be seen, a PDML file generated from the command line tool tshark elicits quite a bit more information (i.e. all of it) from a PCAP than the GUI-based export tools will render. For example, a word count of the .csv file exported from the GUI reveals that it contains 11 lines, versus the PDML file which has 1199. Our tool converts the PDML file into a Weka friendly ARFF, shown (truncated) in Figure 23, making all 1199 (previously unavailable) lines available for datamining in Weka.

Furthermore, the PDML -> pdml2arff.py -> ARFF conversion yields 50 lines of data each with 109 attributes (fields or columns) versus 10 lines, each with 7 attributes from the default export from PCAP to csv provided by Wireshark. This is quite a bit more raw material to work with than before we created our application and ran the PCAP data (PDML) through it.

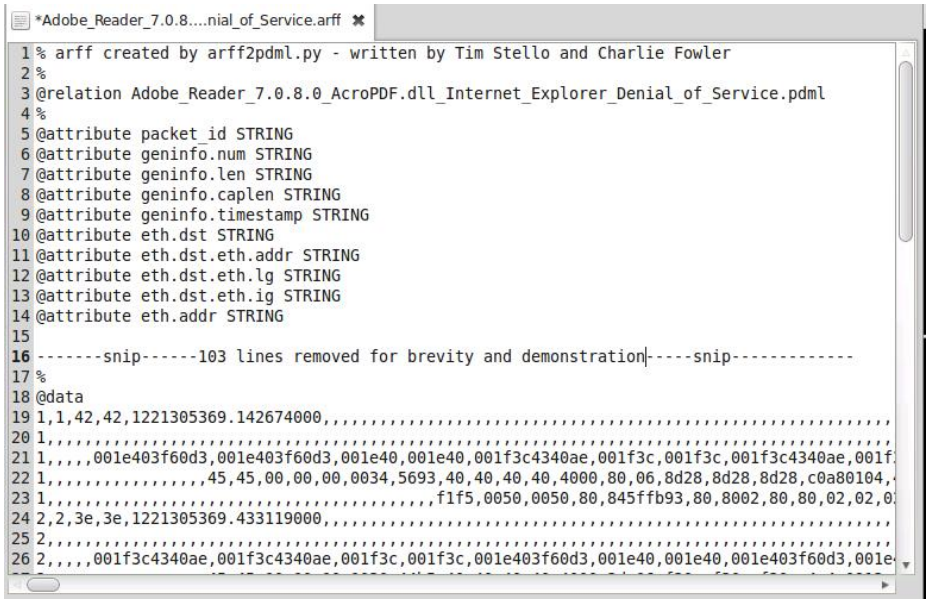

Figure 23: Pdml2arff.py output (truncated)

3) *Pdml2arff output in Weka*

Figure 24 shows the Weka explorer having loaded the ARFF created by our application, pdml2arff.py.

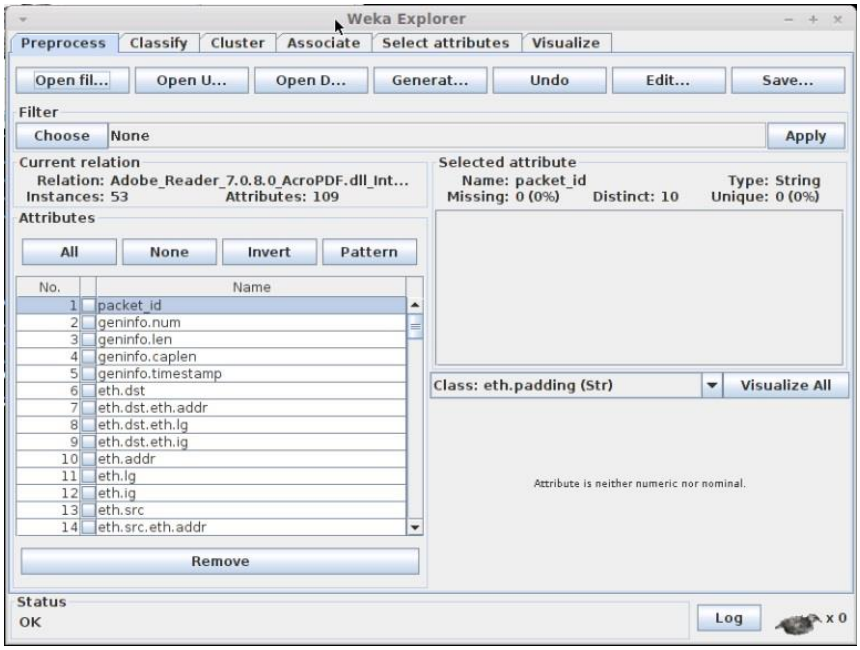

Figure 24: Weka loaded with ARFF from pdml2arff

Figure 25 and Figure 26 show Weka output after running the pdml2arff generated ARFF file through a J48 classifier, and then a SimpleKMeans clustering algorithm respectively. While these particular results do not have much bearing on the strengths or weaknesses of the algorithms, since converting type “string” to “nominal” was the only preprocessing performed on the data, they do demonstrate that the full PCAP can be loaded and operated upon as output from our program.

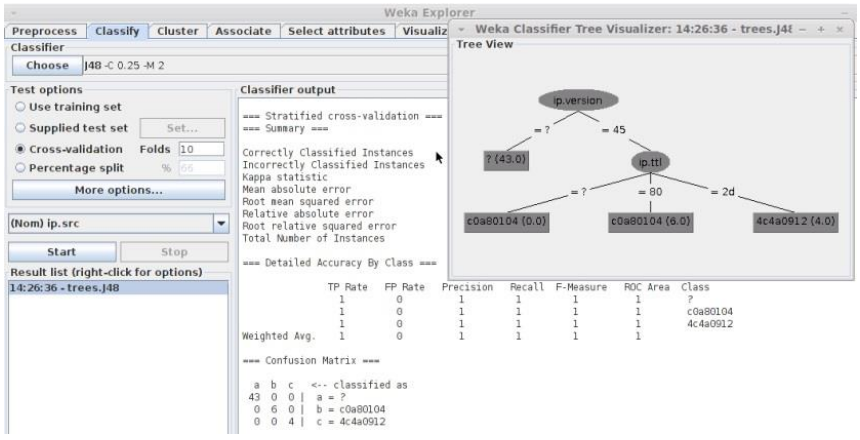

Figure 25: Weka J48 classifier results using pdml2arff generated ARFF file.

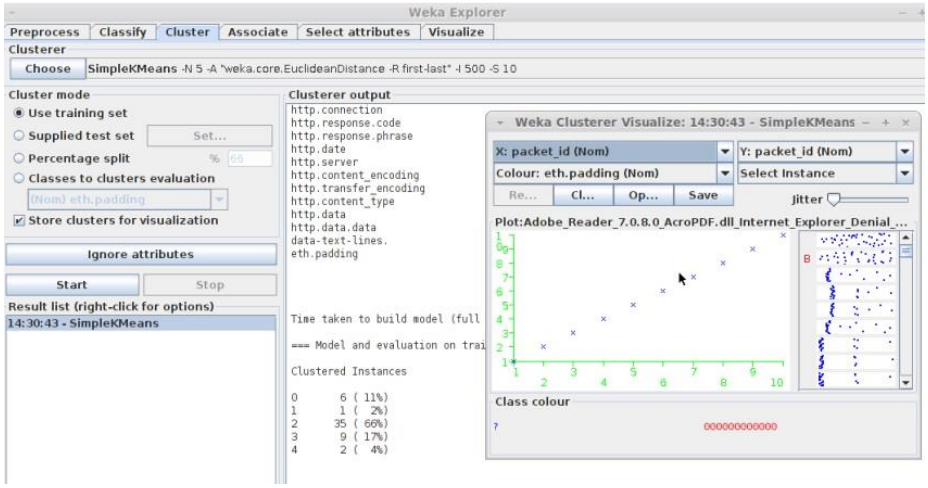

Figure 26: Weka SimpleKMeans results using pdml2arff generated ARFF file.

B. Datamining Results

1) *Classifiers*

When examining the results of the classification runs, it became evident that classifiers which were tested, J48 and NaiveBayes, did an exceedingly good job of spotting well-defined traffic for which they had been adequately trained, but they did not identify undesirable behavior which eludes easy classification. For instance, poring through the run result logs (Appendix B - Run Results) it becomes quickly evident that both J48 and NaiveBayes correctly predicted the classification of *normal* and *malicious* traffic 100% of the time when it matched their rule set, but they did not do well at all identifying scanning traffic in test data (0%, as discussed in greater detail below). A visualization of the simple rule-set created by the J48 classifier following Markey's [48] procedures can be seen in Figure 27.

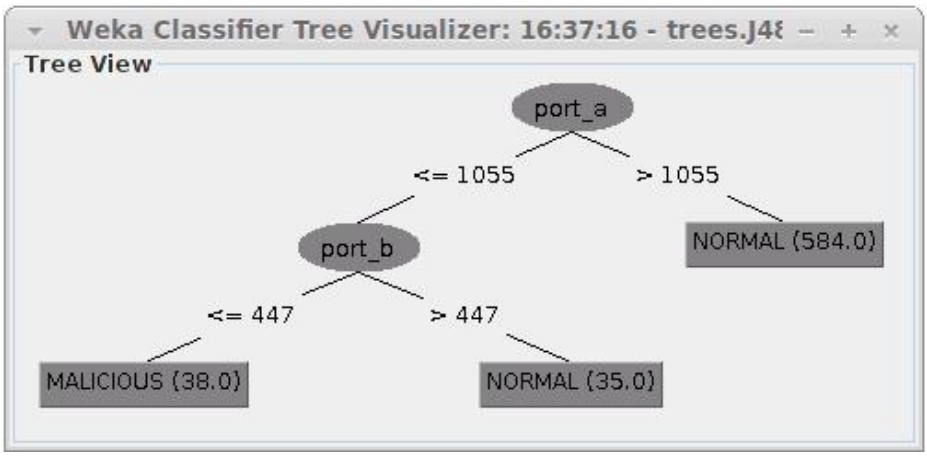

Figure 27: J48 rule visualization

This image graphically displays the decision tree created by the J48 classifier from the pre-classified (training) data. The algorithm ascertained that any tcp network traffic emanating from port_a which is greater than port 1055 is automatically considered to be "NORMAL" traffic. If the traffic runs across ports equal to or below tcp port 1055 it warrants further consideration, and if it is found to be less than or equal to port 447, then the traffic is considered to be "MALICIOUS," otherwise, it is "NORMAL." These rules are accurate given the data with which the classifier was trained.

Classifiers vs. scanning traffic

Despite the classifiers' success at finding known, well-defined exploits, both algorithms (J48 and NaiveBayes) completely failed to identify either horizontal or vertical scans which appeared in test data, even after the classifier(s) had been trained with scanning traffic. Rules describing scanning, in a classification sense, are difficult to craft, since scanning is more of a type of *behavior*, rather than a type of traffic. This is shown graphically in Figure 28 and Figure 29. The first figure shows the training run results, wherein everything is correctly classified, but only because the packets were labeled as such for training. Each clump is found where it should be, at their intersection of the x and y vertices. For the scanning traffic, the relevant packets from a

vertical scan were labeled as “SCANNING”, and the classifier built its rules off of that foreknowledge handed to it through the pre-classified packets. This puts classifiers in good stead as identifiers of known and easily quantified and qualified errant traffic on networks, but also displayed their weaknesses, leaving room for the other types of algorithms pick up where classification successes left off.

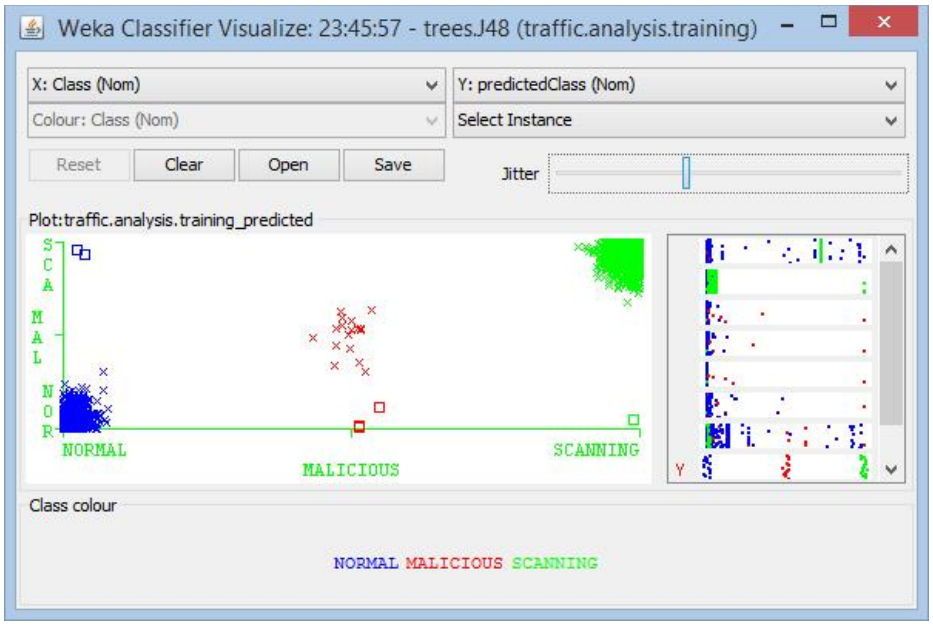

Figure 28: J48 scatter plot - training set

As discussed in the Background section, classification *test data* is typically unlabeled, but in efforts to validate the efficiency of the algorithm, or more precisely, to demonstrate its weaknesses, a set of test data was labeled in order to more obviously bear out a point. Figure 29 shows that in the labeled test set (typically unlabeled data), as with the training set (always labeled data), the blue clump at the bottom left represents all of the instances which were (correctly) classified as *normal* traffic and the middle clump of red x’s show that all of the *malicious* traffic was also correctly classified as *malicious*. However, the clump of green in the lower right hand corner are packets which were actually scanning traffic (as shown by the label reflected directly below the pile) from a PCAP of another vertical scan. This traffic was

incorrectly classified as “NORMAL”, as shown, being aligned with that label at the lower left of the X axis.

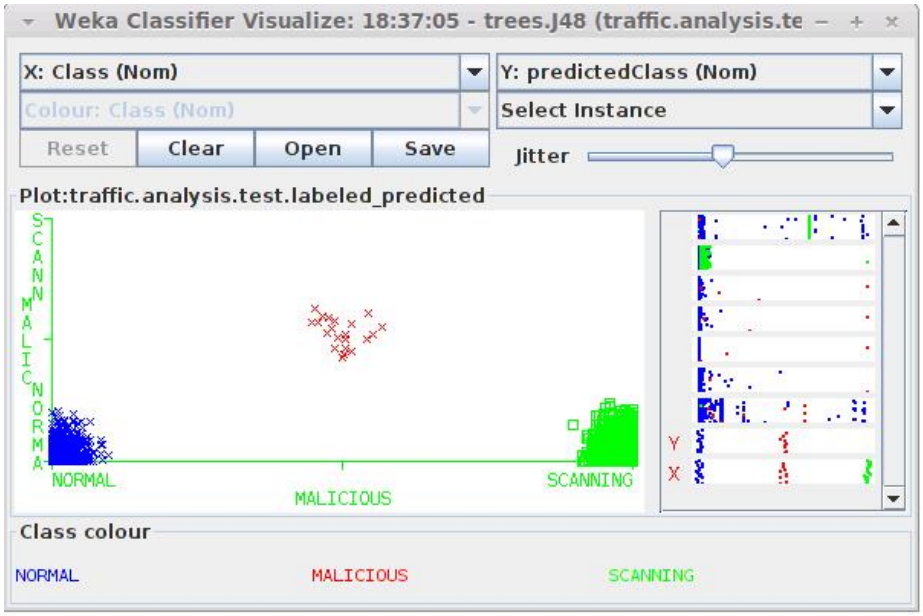

Figure 29: J48 scatter plot - scanning misidentified as normal

This observation is also borne out by Weka’s textual output in lines 641-2642 (also found in Appendix B), where instances labeled as SCANNING were classified as NORMAL:

inst#, actual, predicted, error, probability distribution

1 1:NORMAL 1:NORMAL *1 0 0

2 1:NORMAL 1:NORMAL *1 0 0

3 1:NORMAL 1:NORMAL *1 0 0

----------truncated----------

617 1:NORMAL 1:NORMAL *1 0 0

618 1:NORMAL 1:NORMAL *1 0 0

619 1:NORMAL 1:NORMAL *1 0 0

620 2:MALICIOU 2:MALICIOU 0 *1 0

621 2:MALICIOU 2:MALICIOU 0 *1 0

622 2:MALICIOU 2:MALICIOU 0 *1 0

----------truncated----------

638 2:MALICIOU 2:MALICIOU 0 *1 0

639 2:MALICIOU 2:MALICIOU 0 *1 0

640 2:MALICIOU 2:MALICIOU 0 *1 0

641 3:SCANNING 1:NORMAL + *1 0 0

642 3:SCANNING 1:NORMAL + *1 0 0

643 3:SCANNING 1:NORMAL + *1 0 0

----------truncated----------

2640 3:SCANNING 1:NORMAL + *1 0 0

2641 3:SCANNING 1:NORMAL + *1 0 0

2642 3:SCANNING 1:NORMAL + *1 0 0

For the record, the NaiveBayes algorithm performed with no consequential differences from J48 in these experiments.

2) ***Clusterers***

Clustering vs. scanning

We had postulated that clustering algorithms would readily show interesting communication between hosts, specifically chatter that exceeded "normal" communication, of the type which one would expect from a host which was performing either horizontal or vertical scans. As the figures below demonstrate, this did indeed turn out to be the case. Figure 30 shows just how clearly a vertical scan pops out visually as a cluster. There are a number of items which can be chosen for the X and Y axes, and in every viable configuration, vertical scans shows up very distinctly, as either a ball (Figure 30), or a clear horizontal or vertical bar (as expected).

Additionally, the algorithm was run a number of times, being initialized with up to a hundred clusters which resulted in essentially no change in the results or their visualization. The only perceivable change manifested in some of the background chatter being thrown into their own clusters (seen at the bottom left and halfway up the vertical axis), which exhibited no marked difference in the visualization.

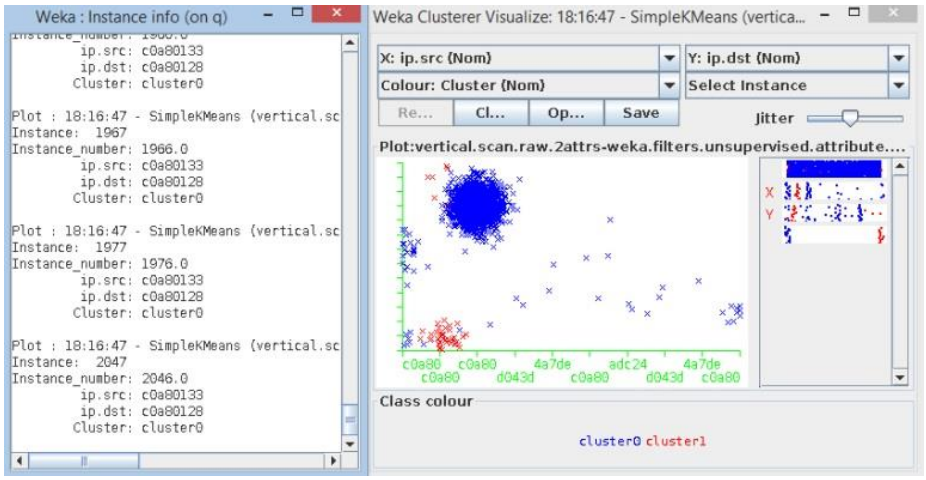

Figure 30: SimpleKMeans – PCAP seeded with a scan, results

Clusterer - unseeded & seeded data

Figure 31 shows the results of the SimpleKMeans algorithm as it performed on the *normal* traffic PCAPs from our baseline set (example.com data), by itself on the left, vs the normal traffic combined with the seeded PCAP (vertical scan) on the right. In the run without the vertical scan (left hand side) the algorithm took all 100 cluster seeds it was passed and, as best as it could, evenly divided up the "normal" traffic amongst them. Interestingly, the same algorithm, seeded with 10 clusters, only used two of them upon seeing the vertical scan, vs the normal traffic. In other words, Figure 31 clearly demonstrates that regardless of how the clustering algorithm's settings are tweaked, a vertical scan will clearly pop out from the traffic when run through the SimpleKMeans algorithm.

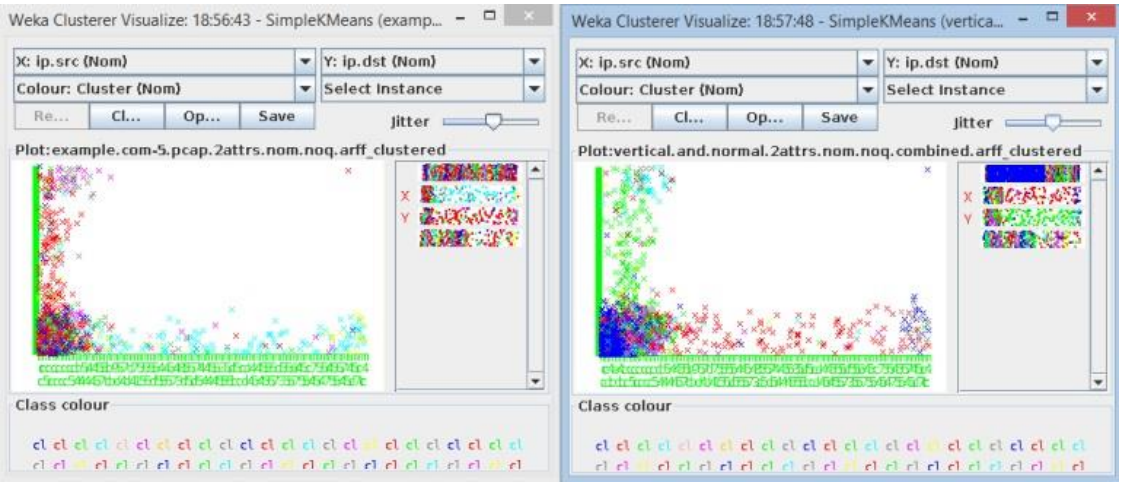

Figure 31: SimpleKMeans: normal traffic, and normal traffic with embedded vertical scan

For all practical purposes, the model and visualization results from the Cobweb clustering algorithm runs on the same data sets, "normal traffic" and "normal traffic seeded with a vertical scan," were identical to the SimpleKMeans results.

Clusterers – horizontal scan

In the following iterations we had the clustering algorithms examine data from the horizontal.scan.raw.pcap which is seeded with a horizontal scan. In the first example, the PCAP was filtered to only show packets in which the "perpetrator" appeared in either the ip.src or ip.dst fields. As can be seen in Figure 32 the Cobweb algorithm nicely separated the one to many conversations initiated by the scanner (ip nominalized to c0a80133 in this PCAP). Unlike SimpleKMeans, the Cobweb algorithm determines on its own how many clusters are necessary to characterize the data, in this case it found 7. When the SimpleKMeans algorithm was run on the same data and seeded with 7 clusters the results were essentially identical.

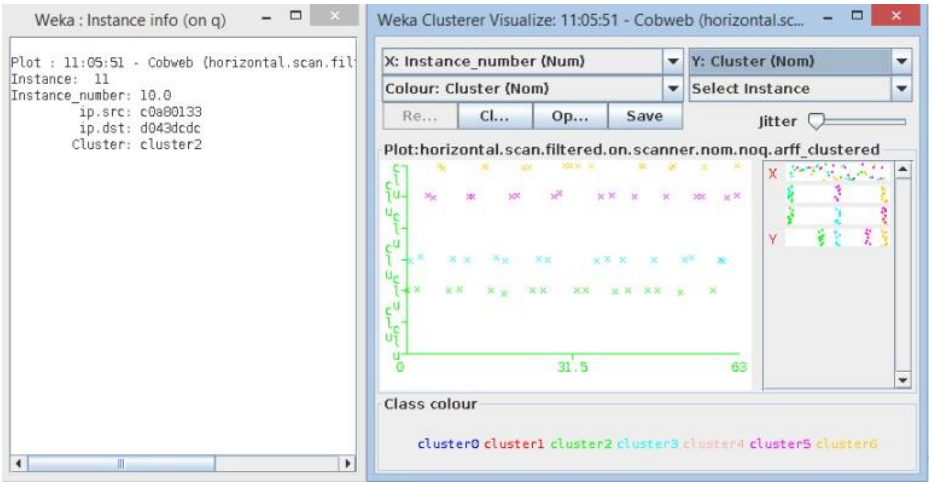

Figure 32: Cobweb horizontal scan, filtered on scanner

As seen in Figure 33, the footprint of a horizontal scan, characterized above, pops out fairly cleanly in the visualization from a SimpleKMeans run against data seeded with this pernicious activity. The other clusters (sets of x's) represent other traffic traversing the network at the same time as the horizontal scan was being performed.

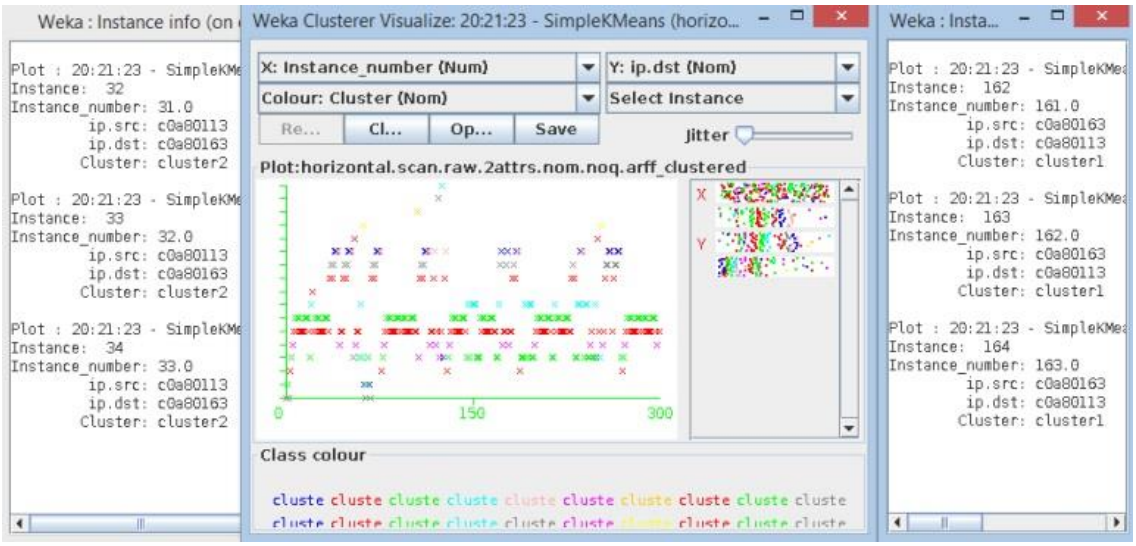

Figure 33: SimpleKMeans - horizontal scan

The red and green bands across the lower half of the image show the call and response between the scanner and its targets. The text windows on the wings of Figure 33 provide insight into this behavior, identifying the nominalized ip address of c0a80113 as the attacker in this data.

Clusterers – seeded malicious data

The clusterers noticed nothing out of the ordinary (read: malicious) when they were given the same data set that was fed into the classifiers which contained "normal traffic" seeded with "malicious traffic." To them it looked just like normal communications between hosts.

3) ***Associator***

In their book Datamining - Practical Machine Learning Tools and Techniques, Witten et al. accurately sum up working with Weka's Apriori algorithm as follows, "As you will discover, it can be challenging to extract useful information using this algorithm." [20] Even so, the results of the experiments are presented below.

Apriori – anomaly detector

The Apriori algorithm demonstrates its strength in the area wherein the others tend to fail. Where clusterers and classifiers elucidate misuse based attacks, Apriori shines in the arena of anomaly detection [131]. It comprehensively, if not cogently, describes the data upon which it operates by identifying items within the data which are frequently found together and provides a confidence level for the rules it generates. It provides an all-important baseline for what "normal" traffic on a given network should look like. Furthermore, some researchers [74] [132] have turned the algorithm on its head, searching for *infrequent* itemsets with high confidence levels, rather than those with *frequent* itemsets. When compared against an already established standard of well-defined behavior, this approach should prove very useful at flagging indicators of likely anomalous behavior.

Apriori - results

In this exercise the Apriori algorithm performed exactly as expected, which was simply to define a characterization of the data. It identified three large itemsets in 18 cycles from the 640

instances in the "normal traffic" PCAPs which were combined (example.com-X.pcap files enumerated above), and then created four rules associating those itemsets as seen in Figure 34. For anomaly based intrusion detection this provides a characterization against which future traffic can be compared.

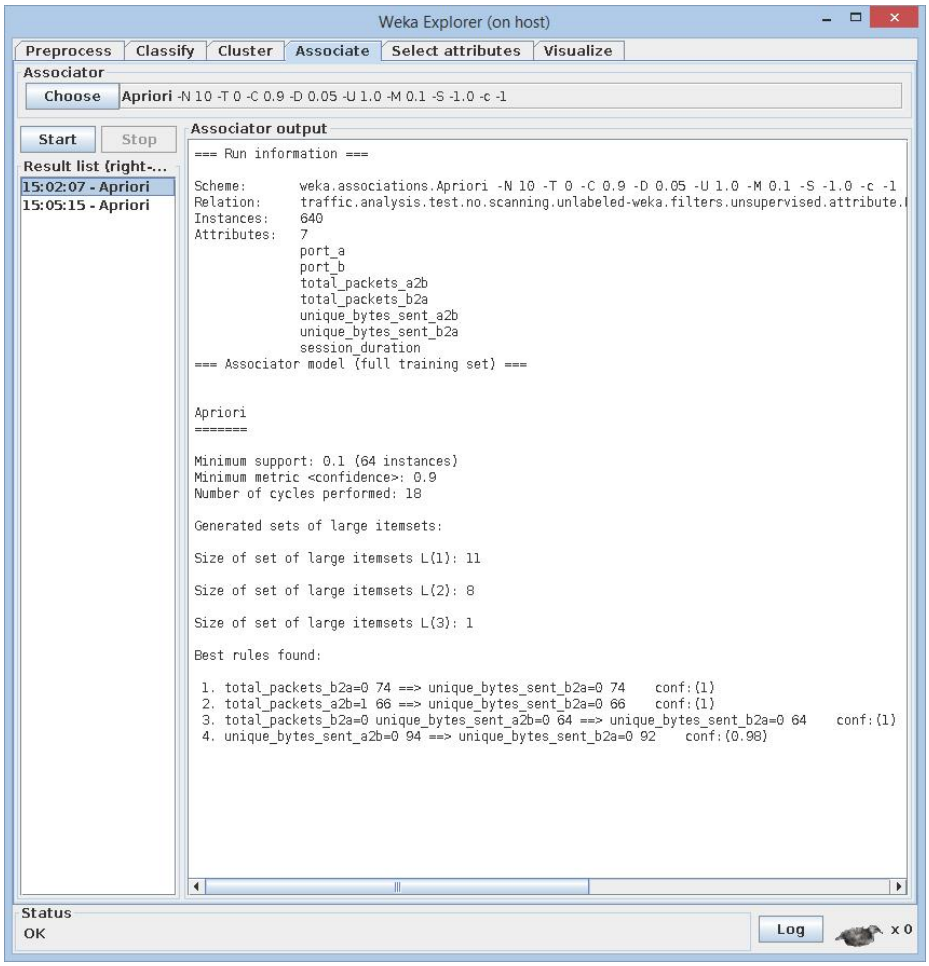

Figure 34: Apriori baseline results

Malicious and scanning data were then added from the relevant PCAPs outlined earlier in the Approach section. As can be seen in Figure 35, the addition of this data brought the number of instances to 2642, from which four large itemsets were identified and 10 rules were generated in five cycles.

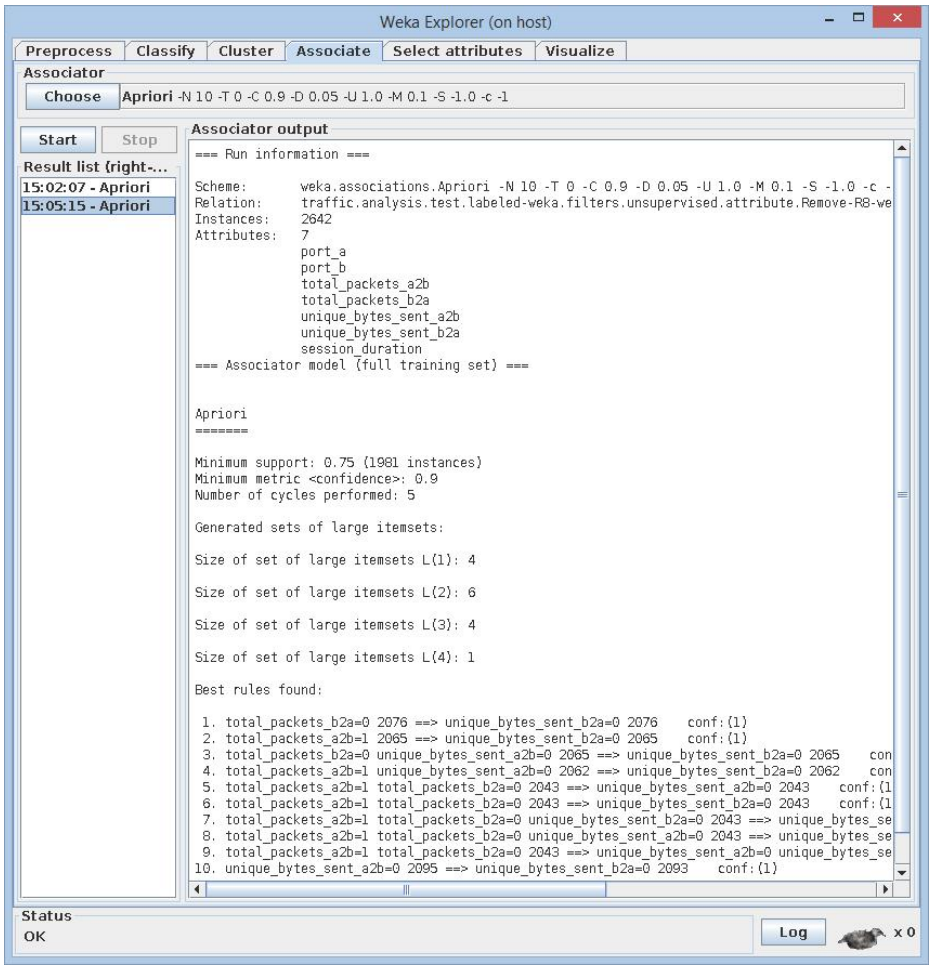

Figure 35: Apriori anomaly results

Although the bytecounts vary, rules 1, 2, and 3 match between the runs, and rule 4 in the baseline finds its match in rule 10 of the second run.

Baseline - best rules found:

1. total_packets_b2a=0 74 ==> unique_bytes_sent_b2a=0 74 conf:(1)
2. total_packets_a2b=1 66 ==> unique_bytes_sent_b2a=0 66 conf:(1)
3. total_packets_b2a=0 unique_bytes_sent_a2b=0 64 ==> unique_bytes_sent_b2a=0 64 conf:(1)
4. unique_bytes_sent_a2b=0 94 ==> unique_bytes_sent_b2a=0 92 conf:(0.98)

Anomalies added in – best rules found:

1. total_packets_b2a=0 2076 ==> unique_bytes_sent_b2a=0 2076 conf:(1)
2. total_packets_a2b=1 2065 ==> unique_bytes_sent_b2a=0 2065 conf:(1)
3. total_packets_b2a=0 unique_bytes_sent_a2b=0 2065 ==> unique_bytes_sent_b2a=0 2065 conf:(1)
4. total_packets_a2b=1 unique_bytes_sent_a2b=0 2062 ==> unique_bytes_sent_b2a=0 2062 conf:(1)
5. total_packets_a2b=1 total_packets_b2a=0 2043 ==> unique_bytes_sent_a2b=0 2043 conf:(1)

6. total_packets_a2b=1 total_packets_b2a=0 2043 ==> unique_bytes_sent_b2a=0 2043 conf:(1)
7. total_packets_a2b=1 total_packets_b2a=0 unique_bytes_sent_b2a=0 2043 ==> unique_bytes_sent_a2b=0 2043 conf:(1)
8. total_packets_a2b=1 total_packets_b2a=0 unique_bytes_sent_a2b=0 2043 ==> unique_bytes_sent_b2a=0 2043 conf:(1)
9. total_packets_a2b=1 total_packets_b2a=0 2043 ==> unique_bytes_sent_a2b=0 unique_bytes_sent_b2a=0 2043 conf:(1)
10. unique_bytes_sent_a2b=0 2095 ==> unique_bytes_sent_b2a=0 2093 conf:(1)

These new, additional rules correlate to traffic that is different from the original baseline and would serve as a flag to a security professional warranting further investigation of the traffic.

C. HI/MAS results

As anticlimactic as it may seem, this particular section in and of itself is fairly thin. As they should be, the results of the HI/MAS were identical to those from the individual runs committed by hand and documented earlier in this chapter. Of course, the HI/MAS results were rendered much more quickly, being done in parallel by the agents, without a GUI or waiting on a human for input or feedback. For example, zeus-sample-2.pcap was processed through 5 algorithms in 11 seconds, while example.com-5.pcap took only 7 seconds to be run through the same set of algorithms. In contrast, doing all of the same steps by hand can take between 5 and 10 minutes per algorithm/run, multiplied by 5 equates to close to, or over half an hour by any measure.

However, the paradigm shift of the HI/MAS does not show up in increased speed or even in the output files which appear at the end of the datamining lanes themselves. Instead, the magnitude of the benefit of this system is felt in the dynamic process and interactive experience afforded by the system in arriving at those results, given its ability to test tweaks quickly across a broad spectrum, and grow the system to suit one's datamining and intrusion detection needs. The bulk of the benefits of this system are outlined in the Approach section and also in the Analysis and Conclusion section. Having said all of that, Figure 36 shows the textual output from each of the five algorithms: a. NaiveBayes classifier, b. SimpleKMeans clusterer, c. Cobweb clusterer, d.

Apriori associator and e. J48 classifier. Any observations or commentary concerning the output can be found earlier in the Results section correlating with the commensurate algorithm.

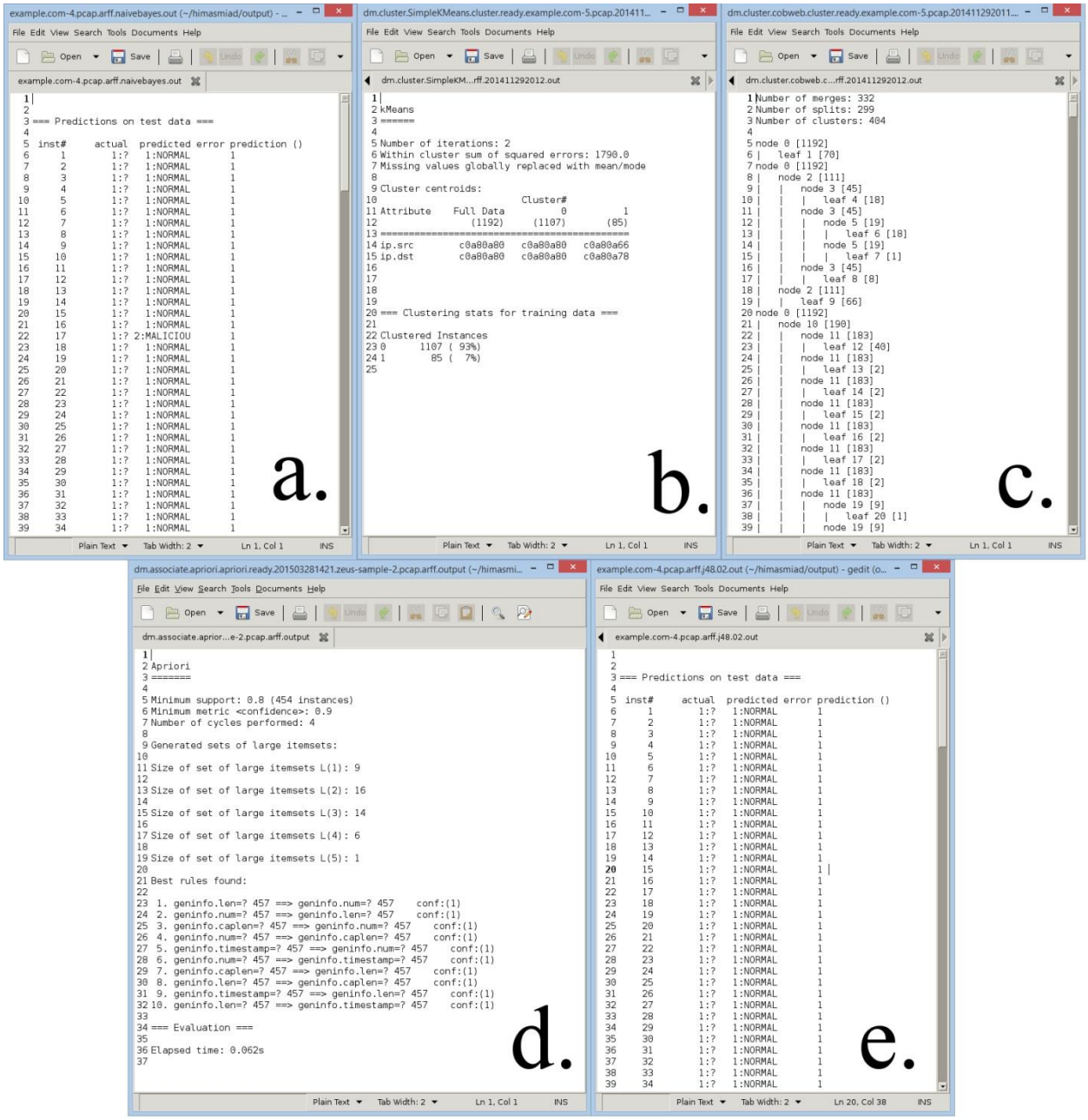

Figure 36: HI/MAS output files

The clustering also outputs *.model files which are binary, so are not displayed in Figure 36.

D. Chapter Summary

In this section we examined the results of the work done grappling with the unexpected hurdle we faced early on in our research, that being one of file conversion from PCAP to ARFF. We then detailed the results from passing the same PCAPs down the various data mining runs and then made some observations regarding the performance of the HI/MAS.

V. ANALYSIS AND CONCLUSIONS

Covered in this chapter…

- Bearing of the results on the thesis
- Measures of success revisited
- A new tool
- Observations from the datamining results

The focus of this research, distilled down to one question, was to determine if a hybrid intelligence/multi-agent (HI/MAS) datamining system would yield better insight into intrusion detection data (PCAPs) than any single algorithm acting on its own. In our efforts to prove or disprove this, a number of interesting insights bubbled to the surface, some of which were expected and some not.

The results of this research, discussed in detail below, laid bare the drawbacks of relying on a single datamining algorithm (or even one type) in extracting intrusion detection related knowledge from PCAPs. They also demonstrated the practical benefits of hybridizing the output of multiple types of algorithms. Furthermore, the repetitive and massive-data oriented natures of datamining tasks lent themselves well to the concepts of automation and division of labor amongst specialized intelligent agents. Once the appropriate questions had been formulated (Abraham's steps 1 and 4a), much of the exhausting work of preprocessing and datamining to find the answers could then be performed by collaborating intelligent agents, which resulted in increased performance and a deeper understanding of knowledge contained in the target data.

A. **Results bear out the thesis**

There is no question that the coping with information overload problem exists; there is also no question that current solutions are unsatisfactory and researchers are invoking unique and new ways to attack the problem [30] [6] [7] [12] [133]. The increase in volume, complexity and lack of homogeneity of data only heightens current systems' painful shortcomings. Having said that, given the level of niche successes that various tools demonstrate, we did not feel it was necessary to completely re-invent the mousetrap. Rather, to reiterate, we believed that the current tools could be teased apart and re-woven together with recent developments in other domains to create a robust, unique and effective problem solving system.

In short, what we discovered in this process was that multiple species or types of knowledge *are* hidden in the layers within our chosen data source (PCAPs), and only by applying several different mining methods on the same data could the various types of relevant information be extracted.

It should also be noted that these tests are but a small set of examples demonstrating what can be accomplished with each type of datamining algorithm on PCAPs. A variety of permutations can be introduced into the processes outlined above to uncover knowledge hiding in the data, for instance, other combinations of attributes can be selected or algorithm settings could be adjusted. Additionally, researchers are constantly producing new or tweaked algorithms which have increased speed, efficacy and accuracy; these could easily be folded into the methodologies outlined herein for even greater yields.

B. New tools for security professionals

1) A HI/MAS

Can data mining accuracy be improved by hybridizing various traditional approaches: Stigmergic insights

In the Literature Review we discussed the insights which can be garnered by observing the behavior of swarms or flocks in nature, and through stigmergy, or the interactions/reactions of these entities to changes in their environment. The bits and bytes swirling around a cyber-attack are no less informative, and can also point to nefarious activities occurring on one's network. This is one of the larger benefits of having a system which simultaneously yields results from multiple algorithms, all of which are looking at different aspects of the same data. Consider for a moment that a hacker is attacking our network using a zero-day exploit. As already noted, any signature based IDS will be completely useless against this exploit, rendered inert since it has no signature for the attack. However, thanks to our HI/MAS we have two other types of algorithms looking at the same data. While all is quiet from the classifiers, the clusterers are continually alerting us that a new conversationalist (intruder) has arrived in the pool of interesting communicating hosts, and probably deserves our attention. Additionally, the associators are telling us, in their own arcane way, that anomalous transactions and communications are occurring, deviations from the baseline. To quote a proverb, "the three strand cord is not easily broken…" [10] and while the one is down, the other two have held fast for us.

With all of this in mind, the answer to the question "is it more accurate" is an emphatic YES! To the author's knowledge, there is no system either on the market or in the research phase which performs as this HI/MAS does, with the benefits that it demonstrates as described above.

What benefits are realized from implementing the processes into a multiagent system?

Serving as a proof of concept the HI/MAS framework we constructed demonstrates the following points of interest in answer to the question above:

- It is entirely feasible, viable, and beneficial to hand over to autonomous agents much of the tedium involved with data mining. It removes the aspect of human error and quick parameter tweaks can be tested quickly and in a repeatable fashion.
- A modular, agent-based system can readily integrate upgraded or different data mining algorithms and their associated preprocessing steps.
- Processing can be spread across multiple hosts for increased performance.
- Autonomous agents collaborating together, yet wielding different types of algorithms do present enhanced knowledge about data contained in a given PCAP (outlined above).

Can datamining speed be increased by hybridizing various traditional approaches?

Following steps similar to those outlined in the Approach section of this paper, a security professional trained in data mining could process an interesting PCAP through all of the datamining algorithms outlined above, and arrive at the same results (albeit one at a time!) that we have using the HI/MAS. Having said that, processing the file serially and manually, even if from the command line, is significantly slower (seconds vs. 30 or more minutes) than sending the file down multiple, parallel lanes, as discussed in the HI/MAS results section earlier. With one stroke of the Enter key, the HI/MAS currently processes one PCAP through 5 algorithms (Apriori, J48, NaiveBayes, SimpleKMeans and Cobweb). That includes all preprocessing, archiving, logging and cleanup. While the automation of tasks in this fashion is nothing new to the computing world in general, an intrusion detection system such as this prototype did not exist prior to this work.

With all of this in mind, not only does the speed of the data mining process (employing a traditional tool – Weka) increase, but the breadth of the area of coverage can many times that of running one algorithm at a time.

2) ***A second new tool for security professionals***

Need for pdml2arff converter restated

Through the efforts to complete our research, to develop a hybrid intelligence/multi-agent datamining system oriented towards intrusion detection [101] [102], an undeniable need for an additional data preprocessing tool was discovered, one that would convert the full contents of a PCAP into ARFF. The literature revealed that the only file or format conversion tools which were extant, either limited in some fashion, or drastically reduced the amount of data extracted out of the PCAP, which was unacceptable to us. The entire contents of a PCAP *can* be converted into PDML, an xml based flat file format. The tool we wrote converts the entire contents of a PDML file into a format suitable for native loading by Weka (ARFF).

Qualifying pdml2arff capabilities

While the current version of our application renders an ARFF which will successfully load into Weka, it has some limitations, mostly imposed by the constraints of Weka itself. Specifically, by default, "missing values" are assigned a value of "?", and all attributes are given the data type of "STRING." Having said that, either the code can be modified to alter this behavior (for instance, changing STRING to NUMERIC) or instances of "?" can be changed to some other representation for "missing values". However, these changes will take effect across the board. If a user requires more granularity, they would need to open the resultant ARFF file and either perform find/replace operations or make adjustments to specific lines using a text editor to further preprocess the data, in order to make it suitable for some of Weka's algorithms.

These preprocessing tasks however, are familiar to and expected by the data miner. In our HI/MAS, each agent can be configured to render the appropriate changes per algorithm programmatically.

C. Measures of success

One of our early potential measures of success sought to answer the question: "Can the speed of datamining be increased with our proposed system?" In short, the answer was "it depends", but further investigation revealed that possibly the wrong question was asked. It should go without saying that adding an extra layer of processing (MAS) on top of an already potentially sluggish system will certainly not increase the speed on that system. On the other hand, quite a bit of speed was imparted to the overall process by having fully automated all of the preprocessing and mining tasks down the three lanes: classify, cluster associate, even with some lanes processing multiple algorithms. Furthermore, by its nature, the multi-agent framework is well suited to having its processes distributed across computing resources. Overall speed would most likely be increased by recasting a monolithic system into the multi-agent paradigm, breaking down and parceling out tasks to simplified, speedy agents, allowing for the distribution and parallelizing of processing, memory usage across multiple resources. Having said that, overall performance would most certainly depend on a variety of factors to include how far-flung the nodes in a distributed system are, network latency, available hardware, virtualization etc.

Another measure of success surrounded the question of whether or not the proposed system would realize improved accuracy or depth of insight into the data and increase knowledge gained from it. As our results show, certainly more sets of eyes and ears are better than one, and the integration of diverse algorithms, playing to the strengths of each certainly yielded more nuggets of knowledge from the data that was tested, than did a single algorithm on its own.

D. Datamining

1) Difficulties/benefits of PCAPs

A number of observations came to light in this particular section of our work. For instance, much of the current and previous research in this field uses system logs and audit files [67], but not data from PCAPs. As it turns out, this is for good reason, the hurdles involved in using PCAPs as intrusion detection data sets for open source datamining are not trivial, as was discovered early on in our research. However, having overcome these roadblocks, after following some not insignificant rabbit trails, this readily available data source proves to be a rich mine of multiple types of information and knowledge. Furthermore, PCAPs are merely the offline format of a packet capture stream, meaning that our methods can be readily automated, streamlined and refined to work in real-time.

2) Different algorithms uncovered different knowledge

Each type of algorithm (classifier, clusterer, associator) brought its own unique and efficient method of extracting knowledge from data. As effective as each algorithm performed individually, this research indicated that when stood up in assembly line fashion, the next algorithm found information left behind by the one before it, and failed to parse out information rendered by the previous one.

The research in this paper demonstrates that one can garner a much more comprehensive and accurate view of what activity is transpiring across any given network by having different types of algorithms working together, each applying their own unique view in examining the same data.

In general, the classifiers performed exceedingly well identifying exploits for which they had been trained but did not do well identifying scanning traffic. Clustering algorithms however,

handily identified both vertical and horizontal type scans between hosts. The Apriori associator fairly characterized normal traffic and sometimes generated new itemsets and confidence rules when anomalies were spotted. This further bolsters our case that some algorithms excel at some tasks, but fall short in others, dropping knowledge on the floor, while multiple algorithms working in tandem, dissecting PCAP data each in their own fashion, produce greater insights and knowledge than any one algorithm on its own.

3) Just scratching the surface

It cannot be restated often enough that these few experiments only begin to scratch the surface of the potential for finding knowledge in PCAPs using the methods described herein. While the learning curve involved with the operation and interpretation of datamining software and its results is not inconsiderable, it is hoped that the procedures outlined will help facilitate bringing these valuable tools into the grasp of network and security administrators in the trenches.

4) Publications resulting from this research

- C. Fowler and R.J. Hammell II, "A Hybrid Intelligence/Multi-Agent System Approach for Mining Information Assurance Data,", *Proceedings of the 9th ACIS International Conference on Software Engineering, Research, Management, & Applications (SERA 2011)*, pp. 169-170, 10-12 August 2011, Baltimore, MD.

- C. Fowler and R.J. Hammell II, "Building Baseline Preprocessed Common Data Sets for Multiple Follow-on Data Mining Algorithms " *Proceedings of the 5th Annual Conference on Information Systems Applied Research (CONISAR 2012),* 1-4 November 2012, New Orleans, LA, h*ttp://proc.conisar.org/2012/pdf/2239.pdf.* **Distinguished PhD Student Paper Award**

- C. Fowler and R.J. Hammell II, "Converting PCAPs into Weka Mineable Data", *Proceedings of the 15th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing* (SNPD 2014), pp. 47-52, June 30-July 2, 2014, Las Vegas, NV.

- C. Fowler and R.J. Hammell II, "Mining Information Assurance Data with a Hybrid Intelligence/Multi-agent System", *Proceedings of the 14h IEEE/ACIS International Conference on Computer and Information Science* (ICIS 2015), June 28-July 1, 2015, Las Vegas, NV, accepted.

E. Chapter Summary

In this section we analyzed the results posted in the previous section, placing them on the backdrop of our expectations outlined in earlier sections. We not only revealed the discovery that our results did bear out our initial thesis, but also discussed lessons learned from some of the unexpected hurdles we encountered along the way. Additionally we enumerated a number of unexpected but useful observations uncovered during the process.

VI. RECOMMENDATIONS FOR FUTURE WORK

Covered in this chapter…

- Incorporation of different data sources
- Expanded use of multi-agent system benefits
- Addition of other algorithms
- Portability to other domains
- Uber layer

A. Different Data Sources

Future endeavors incorporating the traditional and more readily available data sources such as audit and log files, could also build upon the main foundations of this work (multi-algorithm mining) potentially adding even more facets of insight.

B. Take further advantage of multi-agent system benefits

Our hybrid intelligence/multi-agent system provides a sample framework upon which future systems can be built in an effort to better secure our networks against current and future malefactors. As outlined in the Literature Review, a multi-agent system offers a host of benefits only a few of which were implemented in this body of work. Due in part to its modularity, agent technology has the flexibility to quickly add new agents with new functionality as novel threats crop up.

Other researchers are also pursuing approaches similar to, or variations of, a multi-agent system, hybrid intelligence approach for intrusion detection, for instance: using a combination of classifiers [134], examining packet headers *and* audit log data [135], and host-based human immune system and mobile agent paradigms [35]. The field is wide open for the innovation and implementation of these new vectors.

C. Addition of other algorithms

As demonstrated, the agents can be programmed to perform the multitudinous tasks associated with preprocessing seeded or known PCAPs (training data) for use with the different datamining algorithms. They can then run that data through the algorithms in order to train them and/or create models as necessary. The agents can then preprocess PCAPs with unknown traffic (test data) which they can then feed to Weka which should generate results similar to those outlined in this paper, but in an automated fashion.

For instance, Simon et al [136] contribute work using RIPPER, a classification algorithm, reporting good results for scanning traffic. It should be noted however, that in their pre-processing, they manually pre-cluster data, creating Source IP/Destination Pairs (SIDPs). In future research it would be interesting to somehow serialize algorithms, for instance, handing off the SIDP creation process to a clustering algorithm which proved naturally suited to do that, then pass those to the RIPPER classification algorithm for further processing.

Other algorithms which would significantly complement this system are other hybridized approaches like multi-stage cluster sampling and ensemble methods.

D. Portability to other domains

Although our research focused on intrusion detection data, many, if not all of the steps throughout the process are very similar in any domain that seeks knowledge from the large amounts of data they generate. They would also benefit from implementing the processes and procedures outlined herein, with or without coding them into a multi-agent system.

E. Uber (over) layer

Although it would come with a large price tag measured in effort and resources, a worthwhile and sizeable project in the future would be to construct an uber, or over-layer of sorts.

Potentially, this layer could provide a number of functions including a feedback interface and visualization.

A feedback loop to the underlying objects, would allow humans or other entities to interact with the system, broadening or narrowing their scope, or tweaking any of the myriad agents' or datamining algorithms' settings.

As mentioned in the Literature Review, one benefit of multi-agent systems is "emergent behavior" resulting from stigmergy, wherein patterns develop and pop out, offering answers to questions the knowledge seeker did not even know to ask. This requires a visualization of the agents activities leading to the benefits discussed above.

VII. WORKS CITED

[1] J. Gantz, A. Boyd and S. Dowling, "Cutting the clutter: tackling information overload at the source," Xerox, 2009.

[2] T. Segaran, Programming Collective Intelligence, Sebastopol: O'Reilly, 2007.

[3] C. Grosan, A. Abraham and M. Chis, "Swarm Intelligence in Data Mining," *Studies in Computational Intelligence*, pp. 1-20, 2006.

[4] M. Mohammad, N. Sulaiman and O. Muhsin, "A Novel Intrusion Detection System by using Intelligent Data Mining in Weka Environment," *Procedia Computer Science*, vol. 3, pp. 1237-1242, 2011.

[5] Z. Zhu, W. Song and J. Gu, "A Multi-agent and Data Mining Model for TCM Cases Knowledge Discovery," in *Proceedings of the ISECS International Colloquium on Computing, Control and Management*, 2008.

[6] M. Habib, A. E.-H. Sallam and O. Badawy, "Quantitative Association Rule Mining Using a Hybrid PSO/ACO Algorithm (PSO/ACO-AR)," in *Proceedings of the Arab Conference on Information Technology*, 2008.

[7] M. Settles and M. Rylander, "Neural network learning using particle swarm optimizers," *Advances in Information Science and Soft Computing*, pp. 224-226, 2002.

[8] Netflix.com, "Netflix Prize," [Online]. Available: http://www.netflixprize.com/. [Accessed 02 2015].

[9] "Microsoft Malware Classification Challenge (BIG 2015)," [Online]. Available: https://www.kaggle.com/c/malware-classification. [Accessed 02 2015].

[10] t. W. Solomon, "biblegateway.com," [Online]. Available: https://www.biblegateway.com/passage/?search=Ecclesiastes%204:12&version=CJB. [Accessed 11 01 2015].

[11] A. Abraham, "Hybrid Intelligent Systems: Evolving Intelligence in Hierarchical Layers," *Studies in Fuzziness and Soft Computing,* vol. 173, pp. 159-179, 2005.

[12] DeLoach, Oyenan and Matson, "A capabilities-based model for adaptive organizations," *Journal of Autonomous Agents and Multiagent Systems,* vol. 16, no. 1, pp. 13-56, 2008.

[13] L. Cao, C. Luo and C. Zhang, "Agent-Mining Interaction: An Emerging Area," Springer-Verlag, Berlin, 2007.

[14] R. Jayabrabu, V. Saravanan and K. Vivekanandan, "A Framework: Cluster Detection and Multidimensional Visualization of Automated Data Mining Using Intelligent Agents," *International Journal of Artificial Intelligence & Applications,* vol. 3, no. 1, pp. 125-138, 2012.

[15] P. Mitkas, A. Symeonidis, D. Kehagias and I. Athanasiadis, "Application of Data Mining and Intelligent Agent Technologies," *International Journal of Product Lifecycle Management,* vol. 2, pp. 173-186, 2007.

[16] S. Zaidi, S. Abidi and S. Manickam, "Distributed Data Mining From Heterogeneous Healthcare Data Repositories: Towards an Intelligent Agent-Based Framework," in *Proceedings of the 15th IEEE Symposium on Computer Based Medical Systems*, Maribor, 2002.

[17] V. S. Rao, "Multi Agent-Based Distributed Data Mining: An Overview," in *Proceedings of the International Journal of Reviews in Computing*, 2009-2010.

[18] L. Cao, V. Gorodetsky and P. Mitkas, "Agent Mining: The Synergy of Agents and Data Mining," *IEEE Intelligent Systems - Expert Opinion*, pp. 64-72, 2009.

[19] J. Davis and A. Clark, "Data preprocessing for anomaly based network intrusion detection: A review," *Computers & Security*, no. 30, pp. 353-375, 2011.

[20] I. Witten, E. Frank and M. Hall, Data Mining - Practical Machine Learning Tools and Techniques (3rd Ed), Burlington, MA: Morgan Kaufmann Publishers, 2011.

[21] G. Aranda and J. Palanca, "SPADE2," [Online]. Available: https://github.com/javipalanca/spade. [Accessed 31 07 2014].

[22] Kali Linux, "Kali Linux," [Online]. Available: https://www.kali.org/. [Accessed 02 02 2015].

[23] "Ubuntu," [Online]. Available: http://www.ubuntu.com/. [Accessed September 2014].

[24] G. (. Lyon, "Nmap Security Scanner," [Online]. Available: http://insecure.org/. [Accessed 05 09 2014].

[25] S. McCanne, *libpcap: An Architecture and Optimization Methodology for Packet Capture*, 2011.

[26] Wireshark.org, "www.wireshark.org," June 2013. [Online]. Available: http://www.wireshark.org/.

[27] Ponemon Institute, LLC, "2013 Cost of Data Breach Study: Global Analysis," Ponemon Institute - Symantec Corporation, 2013.

[28] M. Phillips, "tcp2d," 18 November 2013. [Online]. Available: https://github.com/mrrrgn/tcp2d.

[29] Weka, "Weka 3 - Data Mining with Open Source Machine Learning Software in Java," 8 April 2011. [Online]. Available: http://www.cs.waikato.ac.nz/~ml/weka/gui_explorer.html .

[30] A. W. Moore and D. Zuev, "Internet Traffic Classification Using Bayesian Analysis Techniques," in *Proceedings of the ACM SIGMETRICS*, Banff, 2005.

[31] T. Stello and C. Fowler, "pdml2arff.py," [Online]. Available: https://github.com/abujed/pdml2arff. [Accessed 16 10 2014].

[32] P. Harrington, Machine Learning in Action, Shelter Island, NY: Manning Publications Co., 2012.

[33] M. Dunham, Data mining introductory and advanced topics, Prentice Hall, 2003.

[34] W. Alsharafat, "Applying artificial neural network and eXtended classifier system for network intrusion detection," *The International Arab Journal of Information Technology,* vol. 10, no. 3, pp. 230-238, 2013.

[35] A. Boukerche, R. Machado, K. Jucá, J. Sobral and M. Notare, "An agent based and biological inspired real-time intrusion detection and security model for computer network operations," *Computer Communications,* vol. 30, no. 13, pp. 2649-2660, 2007.

[36] A. Das and S. Sathya, "Association Rule Mining for KDD Intrusion Detection Data Set," *International Journal of Computer Science and Informatics,* vol. 2, no. 3, 2012.

[37] W. Feng, Q. Zhang, G. Hu and J. Huang, "Mining network data for intrusion detection through combining SVMs with ant colony networks," *Future Generation Computer Systems,* no. 37, pp. 127-140, 2014.

[38] American Banker, "American Banker - Glossary," [Online]. Available: http://www.americanbanker.com/glossary/d.html. [Accessed 2 2 2010].

[39] Guaranteed Sports Pick, [Online]. Available: http://www.guaranteedsportspick.com/page.asp?action=view&SiteID=64. [Accessed 2 2 2010].

[40] T. McKenna, "A Data Warehouser's Vocabulary (Part 2)," [Online]. Available: http://blog.todmeansfox.com/2008/06/13/a-datawarehousers-vocabulary-part-2/. [Accessed 2 2 2010].

[41] Visa Canada, "Visa Canada - Business Glossary," [Online]. Available: www.visa.ca/smallbusiness/businesstools/business-glossary/index.cfm. [Accessed 10 12 2009].

[42] R. L. Grossman, "Data Mining: Challenges and Opportunities for Data Mining the Next Decade," in *Technical Report, Laboratory for Advanced Computing*, Boca Raton, FL, 1999.

[43] A. Jain, M. Murty and P. Flynn, "Data Clustering: A Review," *ACM Computing Surveys*, vol. 31, 1999.

[44] Z. Shi, T. Khoshgoftaar and N. Seliya, "CLUSTERING-BASED NETWORK INTRUSION DETECTION," *International Journal of Reliability, Quality and Safety Engineering*, vol. 14, no. 2, pp. 169-187, 2007.

[45] R. Quinlan, "Is C5.0 Better than C4.5?," [Online]. Available: http://www.rulequest.com/see5-comparison.html.

[46] D. Dankel II, "ID3," [Online]. Available: http://www.cise.ufl.edu/~ddd/cap6635/Fall-97/Short-papers/2.htm.

[47] S. Kotsiantis and D. Kanellopoulos, "Combining bagging, boosting and random subspace ensembles for regression problems," *Internationsn Journal of Innovative Computing, Information and Control*, vol. 8, pp. 3953-3961, 2012.

[48] Markey and Atlasis, ""Using Decision Tree Analysis for Intrusion Detection: A How-to Guide"," Global Information Assurance Certification Paper, 2011.

[49] L. Koc, T. Mazzuchi and S. Sarkani, "A network intrusion detection system based on a hidden naive bayes multiclass classifier," *Expert Systems with Applications,* vol. 39, no. 18, pp. 13492-13500, 2012.

[50] A. Mitrokotsa and C. Dimitrakakis, "Intrusion detection in MANET using classification algorithms: The effects of cost and model selection," *Ad Hoc Networks,* vol. 11, pp. 226-237, 2013.

[51] J. Oberst, "Efficient Data Clustering and How to Groom Fast-Growing Trees," uni-potsdam.de, 2009.

[52] S. Guha, R. Rastogi and K. Shim, "CURE: an efficient clustering algorithm for large databases," in *Proceedings of SIGMOD '98*, New York, NY, 1998.

[53] Z. He, X. Xu, S. Deng and B. Dong, "K-Histograms: An Efficient Clustering Algorithm for Categorical Dataset," Cornell University Library: Computer Science > Artificial Intelligence, 2005.

[54] J. MacQueen, "Some methods for classification and analysis of multivariate observations," in *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability,* Berkeley, CA, 1967.

[55] G. Wilkin and X. Huang, "K-Means Clustering Algorithms: Implementation and Comparison," in *Proceedings of the Second International Multi-Symposium on Computer and Computational Sciences (IMSCCS 2007),* 2007.

[56] A. Moore, "K-means and Hierarchical Clustering," [Online]. Available: http://www.autonlab.org/tutorials/kmeans.html.

[57] S. Garg and R. Jain, "Variations of K-means Algorithm: A Study for High-Dimensional Large Data Sets," *Information Technology Journal*, vol. 5, no. 6, 2006.

[58] S.-C. Chu, J. Roddick and J.-S. Pan, "An Incremental Multi-Centroid, Multi-Run Sampling Scheme for K-medoids-based Algorithms - Extended Report," in *Proceedings of Data Mining III - 3rd International Conference on Data Mining Methods and Databases*, Bologna, Italy, 2002.

[59] M. Zdravko and D. Larose, Data Mining the Web: Uncovering Patterns in Web Content, Structure, and usage, Hoboken, NJ: John Wiley & Sons, 2007.

[60] S. Guha, R. Rastogi and K. Shim, "ROCK: A Robust Clustering Algorithm for Categorical Attributes," in *Proceedings of the 15th International Conference on Data Engineering (ICDE'99)*, 1999.

[61] J. Gehrke, R. Ramakrishnan and V. Ganti, "CACTUS - clustering categorical data using summaries," in *Proceedings of the 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '99)*, 1999.

[62] D. Hebert, "Data Warehousing/Mining - Lecture notes for Chapter 8," [Online]. Available: http://www.cs.tufts.edu/comp/150DW/lectures/DWDesign.ppt. [Accessed 3 2010].

[63] J. Nieves, "Data Clustering for Anomaly Detection in Network," in *Proceedings of the Research Alliance in Math and Science*, Oak Ridge, TN, 2009.

[64] E. Papalexakis, A. Beutel and P. Steenkiste, "Network anomaly detection using co-clustering," in *Proceedings of the 2012 International Conference on Advances in Social Network Analysis andn Mining*, Istanbul, Turkey, 2012.

[65] Y. Anto, The Art of Hacking, LAP LAMBERT Academic Publishing, 2012.

[66] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules in large databases," in *Proceedings of the 20th International Conference on Very Large Data Bases*, Santiago, Chile, 1994.

[67] E. Saboori, S. Parsazad and Y. Sanatkhani, "Automatic Firewall rules generator for Anomaly Detection Systems with Apriori algorithm," in *Proceedings of the CoRR*, 2012.

[68] Microsoft, "Data Mining Algorithms (Analysis Services - Data Mining)," [Online]. Available: http://msdn.microsoft.com/en-us/library/ms175595.aspx.

[69] MADlib, "MADlib:Apriori Algorithm," [Online]. Available: http://doc.madlib.net/v1.0/group__grp__assoc__rules.html. [Accessed 29 August 2014].

[70] J. Han, M. Kamber and J. Pei, Data Mining Concepts and Techniques (Third Edition), Waltham, MA: Elsevier, 2012.

[71] C. Gyorodi, R. Gyorodi and S. Holban, "A Comparative Study of Association Rules Mining Algorithms," in *Proceedings of the 1st Romanian-Hungarian Joint Symposium on Applied Computational Intelligence*, Timisoara, Romania, 2004.

[72] D. Jeya, "Eclat algorithm in association rule mining," [Online]. Available: http://www.slideshare.net/deepa15/eclat-37310304. [Accessed 26 03 2015].

[73] G. Florez, S. Bridges and R. Vaughn, "An Improved Algorithm for Fuzzy Data Mining," in *Proceedings of the Fuzzy Information Processing Society, 2002. Proceedings. NAFIPS. 2002 Annual Meeting of the North American*, 2002.

[74] A. Rahman, C. Ezeife and A. Aggarwal, "WiFi Miner: An Online Apriori-Infrequent Based Wireless Intrusion Detection System," in *Knowledge Discovery from Sensor Data*, Berlin, Springer, 2010, pp. 76-93.

[75] Google inc., "Google," [Online]. Available: http://www.google.com. [Accessed 12 2014].

[76] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," in *Proceedings of the OSDI '04: Sixth Symposium on Operating System Design and Implementation*, San Francisco, 2004.

[77] Google Code University, "Advanced sections - MapReduce and HDFS," [Online]. Available: http://hadooptutorial.wikispaces.com/Advanced+sections+-+MapReduce+and+HDFS.

[78] Amazon.com, "Amazon," [Online]. Available: http://www.amazon.com. [Accessed 12 2014].

[79] Amazon Web Services, "Amazon Elastic MapReduce," [Online]. Available: http://aws.amazon.com/elasticmapreduce/.

[80] Yahoo!, "Yahoo!," Yahoo! inc., [Online]. Available: http://www.yahoo.com. [Accessed 12 2014].

[81] Yahoo!, "Hadoop!," Yahoo!, [Online]. Available: http://hadoop.apache.org/. [Accessed 12 2014].

[82] Apache.org, "Apache.org," [Online]. Available: http://www.apache.org. [Accessed 12 2014].

[83] D. DeWitt and M. Stonebraker, "MapReduce: A major step backwards," 2008. [Online]. Available: http://databasecolumn.vertica.com/2008/01/mapreduce-a-major-step-back.html.

[84] Happy Jack Software, "Google's MapReduce," [Online]. Available: http://www.happyjacksoftware.com/techtalk_pdfs/MapReduce.pdf.

[85] A. J. Deepa and V. Kavitha, "A Comprehensive Survey on Approaches to Intrusion Detection System," in *Proceedings of the International Conference on modeling optimisation and computing*, 2012.

[86] T. Holland, "Understanding IPS and IDS: Using IPS and IDS together for Defense in Depth," SANS Institute, 2004.

[87] Tippit, Inc., "IDS IPS Buyer's Guide - IT Security," www.itsecurity.com, 2007.

[88] M. Heron, V. Hanson and I. Ricketts, "Open source and accessibility: advantages and limitations," *Journal of Interaction Science,* 2013.

[89] P. &. Pfleeger, Security in Computing - Third Edition, Upper Saddle: Prentice Hall, 2003.

[90] D. Bhattacharyya and J. Kalita, Network Anomaly Detection: A Machine Learning Perspective, Chapman and Hall/CRC, 2013.

[91] A. Youssef and A. Emam, "Network Intrusion Detection Using Data Mining and Network Behaviour Analysis," *International Journal of Computer Science & Information Technology,* vol. 3, no. 6, pp. 87-98, 2011.

[92] P. Nevlud, M. Bures, L. Kapicak and J. Zdralek, "Anomaly-based Network Intrusion Detection Methods," *Advances in Electrical and Electronic Engineering,* pp. 468-474, 2013.

[93] Snort.org, "Snort," Snort.org, [Online]. Available: https://www.snort.org/. [Accessed 23 08 2014].

[94] N. Padhy, P. Mishra and R. Panigrahi, "The Survey of Data Mining Applications and Feature Scope," *International Journal of Computer Science, Engineering and Information Technology,* vol. 2, no. 3, pp. 43-58, 2012.

[95] A. Kapoor, "Data Mining: Past, Present and Future Scenario," *Internationl Journal of Emerging Trends & Technology in Computer Science,* vol. 3, no. 1, pp. 95-99, 2014.

[96] S. Goele and N. Chanana, "Data Mining Trend in Past, Current and Future," in *Proceedings of the I-Society 2012*, Punjab, 2012.

[97] D. Kumar and D. Bhardwaj, "Rise of Data Mining: Current and Future Application Areas," *IJCSI International Journal of Computer Science Issues,* vol. 8, no. 5, pp. 256-260, 2011.

[98] Smita and P. Sharma, "Use of Data Mining in Various Field: A Survey Paper," *IOSR Journal of Computer Engineering (IOSR-JCE),* vol. 16, no. 3, pp. 18-21, 2014.

[99] G. V. Nadiammai and M. Hemalatha, "Effective approach toward Intrusion Detection System using data mining techniques," in *Proceedings of the Egyptian Informatics Journal*, Cairo, 2013.

[100] S. C. Suh, Practical Applications of Data Mining, Jones and Bartlett Learning, 2011.

[101] C. Fowler and R. Hammell II, "A Hybrid Intelligence/Multi-agent System Approach for Mining Information Assurance Data," in *Proceedings of the ACIS International Conference on Software Engineering Research, Management and Applications*, Baltimore, MD, 2011.

[102] C. A. Fowler and R. J. Hammell II, "Building Baseline Preprocessed Common Data Sets for Multiple Follow-on Data Mining Algorithms," in *Proceedings of the Conference on Information Systems Applied Research*, New Orleans, 2012.

[103] S. t. Wise, "Proverbs Chapter 6:6-8," ~950 BC. [Online]. Available: http://www.biblegateway.com/passage/?search=proverbs%206:6-8&version=NIV.

[104] J. Haye, "Blind men and an elephant," [Online]. Available: https://www.a-n.co.uk/blogs/university-campus-suffolk-28/date/2014/01/page/2. [Accessed 9 12 2014].

[105] S. Powers and J. He, "A hybrid artificial immune system and Self Organising Map for network intrusion detection," *Information Sciences,* no. 178, pp. 3024-3042, 2008.

[106] F. Otero, A. Freitas and C. Johnson, "Inducing decision trees with an ant colony optimization algorithm," *Applied Soft Computing,* no. 12, pp. 3615-3626, 2012.

[107] M. Woolridge, An Introduction to MultiAgent Systems, West Sussex: John Wiley & Sons Ltd, 2002.

[108] G. Weiss, Multiagent Systems - A Modern Approach to Distributed Artificial Intelligence, London: The MIT Press, 1999.

[109] I. Dickinson, "Agents Standards," Hewlett Packard, Bristol, 1997.

[110] "The Foundation for Intelligent Physical Agents," [Online]. Available: http://fipa.org/. [Accessed September 2014].

[111] B. Partridge, "Structure and Function of Fish Schools," *Scientific American,* vol. 246, no. 6, p. 114, June 1982.

[112] M. Dorigo, E. Bonabeau and G. Theraulaz, "Ant algorithms and stigmergy," *Future Generation Computer Systems,* pp. 851-871, 2000.

[113] Couzin, Krause, James, Ruxton and Franks, "Collective Memory and Spatial Sorting in Animal Groups," *Journal of Theoretical Biology,* vol. 218, no. 1011, 2002.

[114] Softcomputing.net, "HIS - Hybrid Intelligent Systems web site," [Online]. Available: http://www.softcomputing.net/his/his.html. [Accessed 5 2 2010].

[115] Microsoft, "Microsoft Network Monitor 3.4 (archive)," [Online]. Available: http://www.microsoft.com/en-us/download/details.aspx?id=4865. [Accessed 12 2014].

[116] M. Revathi and T. Ramesh, "Network intrusion detection system using reduced dimensionality," *Indian Journal of Computer Science and Engineering Vol 2, no 1,* 2011.

[117] "Python Software Foundation," [Online]. Available: https://www.python.org/. [Accessed 7 11 2014].

[118] C. Fowler and R. Hammell, "Converting PCAPs into Weka Mineable Data," in *Proceedings of the 15th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD 2014)*, Las Vegas, 2014.

[119] C. Fowler and R. J. Hammell II, "The benefits of using multiple data mining algorithms on a dataset for instrusion detection," in *Proceedings of the International Conference on Computer and Information Science (ICIS 2015)*, Las Vegas, NV, submitted, 2015.

[120] "Honeynet Project, The," [Online]. Available: https://projects.honeynet.org/honeywall/. [Accessed 12 2014].

[121] Evilfingers.com, "PCAPSploits," February 2014. [Online]. Available: https://www.evilfingers.com/repository/pcaps_sploits.php.

[122] M. Parkour, "Cantagio - malware dump," [Online]. Available: http://contagiodump.blogspot.com. [Accessed 17 03 2014].

[123] "OpenPacket," [Online]. Available: http://www.openpacket.org. [Accessed 2 5 2012].

[124] I. Holko, "Anotace NetFlow provozu," 2009.

[125] C. A. C. Pereira, "Uma metodologia de classificacao de trafego http com a netramark," Recife, Brazil, 2012.

[126] G. Ramirez, B. Caswell, N. Rathaus and J. Beale, Nessus, Snort, & Ethereal Power Tools, Rockland, MA: Syngress, 2005, p. 208.

[127] Wireshark, "Wireshark · Display Filter Reference Index," 14 11 2013. [Online]. Available: http://www.wireshark.org/docs/dfref/.

[128] M. Perepletchikov, C. Ryan and K. Frampton, "Comparing the Impact of Service-Oriented and Object-Oriented Paradigms on the Structural Properties of Software," *On the Move to Meaningful Internet Systems 2005: OTM 2005 Workshops*, vol. 3762, pp. 431-441, 2005.

[129] S. Lumetta, *Finite state machine design examples, Part I*, 2012.

[130] R. M. Bastos and J. P. M. de Oliveira, "A conceptual modeling framework for multi-agent information systems," in *Proceedings of the 19th International Conference on Conceptual Modeling, ER'00*, Berlin, 2000.

[131] K. Nalavade and B. Meshram, "Finding Frequent Itemsets using Apriori Algorithm to Detect Intrusions in Large Dataset," *International Journal of Computer Applications & Information Technology*, vol. 6, no. 1, pp. 84-92, 2014.

[132] S. Kamepalli, R. Kurra and S. Krishna, "Apriori Based: Mining Infrequent and Non-Present Item Sets from Transactional Data Bases," *International Journal of Electrical & Computer Science*, vol. 14, no. 3, pp. 31-35, 2014.

[133] J. Zhang and M. Zulkernine, "Anomaly Based Network Intrusion Detection with Unsupervised Outlier Detection," in *Proceedings of the IEEE International Conference on Communications*, 2006.

[134] M. Panda, A. Abraham and M. Patra, "A Hybrid Intelligent Approach for Network Intrusion Detection," in *Proceedings of the International Conference on Communication Technology and System Design 2011*, 2012.

[135] W. Jianping, C. Min and W. Xianwen, "A Novel Network Attack Audit System based on Multi-Agent Technology," in *Proceedings of the International Conference on Solid State Devices and Materials Science*, 2012.

[136] G. J. Simon, H. Xiong, E. Eilertson and V. Kumar, "Scan Detection: A Data Mining Approach," in *Proceedings of the SDM*, 2006.

[137] C. L. Giles, R. Sun and J. M. Zurada, "Neural Networks and Hybrid Intelligent Models: Foundations, Theory and Applications," *IEEE Transactions on Neural Networks,* vol. 9, no. 5, 1998.

[138] National Institute of Standards and Technology, "NP Complete," 2012. [Online]. Available: http://www.itl.nist.gov/div897/sqg/dads/HTML/npcomplete.html. [Accessed 12 2012].

[139] WP Authors, "PWND!," 2012. [Online]. Available: http://en.wikipedia.org/wiki/PWNED!. [Accessed 12 2012].

[140] Nature.com, "Glossary," 2012. [Online]. Available: http://www.nature.com/nrmicro/journal/v3/n2/glossary/nrmicro1084_glossary.html. [Accessed 12 2012].

[141] WP Authors, "Small-world network," 2012. [Online]. Available: http://en.wikipedia.org/wiki/Small-world_network. [Accessed 12 2012].

[142] WP Authors, "Stigmergy," 2012. [Online]. Available: http://en.wikipedia.org/wiki/Stigmergy. [Accessed 12 2012].

[143] V. Ramos, C. Fernandes and A. Rosa, "Brains, Minds & Media," *Journal of New Media in Neural and Cognitive Science,* 2005.

[144] D. Huynh, N. Jennings and N. Shadbolt, "FIRE: An Integrated Trust and Reputation Model for Open Multi-Agent Systems," in *Proceedings of ECAI 2004: 16th European Conference on Artificial Intelligence*, Amsterdam, 2004.

[145] F. Brazier, B. Dunin-Keplicz, N. Jennings and J. Treur, "DESIRE: Modelling Multi-Agent Systems in a Compositional Formal Framework," *International Journal of Cooperative Information Systems*, vol. 6, no. 1, pp. 67-94, 1997.

[146] WP Authors, "Hadoop," 2012. [Online]. Available: http://en.wikipedia.org/wiki/Hadoop. [Accessed 12 2012].

[147] Repast Software, "Repast," 2012. [Online]. Available: http://repast.sourceforge.net/. [Accessed 12 2012].

[148] M. Seibert, B. Rhodes, N. Bomberger, P. Beane, J. Sroka and W. Kogel, "SeeCoast Port Surveillance," in *Proceedings of SPIE - The International Society for Optical Engineering*, Kissimmee, FL, 2006.

[149] Oak Ridge National Laboratory, "Vipar," 2012. [Online]. Available: http://www.csm.ornl.gov/~v8q/Homepage/Projects/vipar.htm. [Accessed 12 2012].

[150] JADE developers, "Introduction to JADE," 2015. [Online]. Available: http://jade.tilab.com/documentation/tutorials-guides/introduction-to-jade/. [Accessed 04 2015].

[151] Jason Developers, "Jason News," 2015. [Online]. Available: http://sourceforge.net/p/jason/news/. [Accessed 04 2015].

[152] Agentprogramming site authors, "Agent Platforms," [Online]. Available: http://agentprogramming.com/agent-platforms/. [Accessed 04 2015].

VIII. APPENDICES

Covered in the appendices…

Preprocessing scripts

Run results from data mining operations documented in chapter III. Approach

A glossary

A list of acronyms used in intrusion detection and multi-agent systems disciplines

A list of various software packages used in intrusion detection and multi-agent systems disciplines

A. Appendix A - Preprocessing scripts

In this appendix we have documented the various preprocessing scripts which can be used if one wishes to follow along in our research. Many of these scripts were incorporated into our hybrid intelligence/multi-agent system

```
**************************************
**** markeypcapmod for J48 classifier. ****
**************************************

**********start markeypcapmod script**********
#!/bin/bash
# markeypcapmod
# this script requires tcptrace and csvtool to be installed.
```

this takes one argument and it is not the file you want to operate on, it is a file which contains a list of files you want to operate on

this script was written to save my mouse from doing all of the hard work spelled out in Markey and Atlasis' paper:

"Using Decision Tree Analysis for Intrusion Detection: A How-to Guide"

for example, it will take all of the "example.com" files, process them and put them together in filename of your choosing.

this script only does one "class" of files at a time, so use it on all the "normal" files, then change the variables to process

the "malicious" files. To put together the resultant "malicious" and "normal" csv's I do this:

copy one of them to traffic_analysis.csv:

cp normal.csv traffic_analysis.csv

then on the next one:

grep –v port_a malicious.csv >> traffic_analysis.csv

It is spelled out in the comments but… this script will run the tcptrace command with the –csv –l switches on every file

listed in a file (created as below). It then runs the csvtool command, leaving only Markey's interesting columns.

Then it takes those results and puts them all together in one csv file which can be imported by Weka, or

put together with the other class file.

```
# first, in the directory where the files which you wish to process are, do an "ls > <filelist>" and remove any lines that are not
# the files you're trying to operate on, also set your variables in this script.
# Then type: markey.pcapmod <filelist>
# this should process each file (pcap) in the <filelist> and put the results all together in one *.csv
# (spelled out in the variable in this script)  iaw Markey's steps on page 23 of his paper (see above)

MUNGEFILE="malicious.csv" # normalfiles.csv or maliciousfiles.csv or tcp-scan.csv, or, or, or…
BENT="MALICIOUS" # "NORMAL" or "MALICIOUS" or "SCANNING" or "APT" or, or, or…


for i in `cat $1`
        do
        # tcptrace does its magic and dumps the results to a tmp file
        tcptrace --csv -l $i > $i.tmp
        # remove first 8 lines from the file
        sed -i -e '1,8d' $i.tmp
```

```
# change the name of a column because I could not find “session_duration” in the pcaps, I substituted some other duration column
sed -i 's/idletime_max_a2b/session_duration/g' $i.tmp
# remove the ,,, from this artifact line that somehow showed up (could probably do this all in one line - see next line)
sed -i '/,,,,/d' $i.tmp
# remove blank lines (could probably have matched the ,,,, and deleted all in one line, but this is helpful to know too :)
sed -i '/^$/d' $i.tmp
# csvtool yanks out only the columns we want
csvtool col 4,5,8,9,22,23,84 $i.tmp >> $i.csv
# append ",Class" to the end of the line that has "port_a" in it - effectively giving us a new column heading
sed -i '/^port_a/ s/$/,Class/' $i.csv
# append ",NORMAL" or "MALICIOUS" (depending on variable above) to every line that does NOT have "port_a" in it
# effectively populating the rows with whatever is in the variable above (i.e. NORMAL) in the new column
sed -i "/^port_a/! s/$/,$BENT/" $i.csv
# cleanup
rm $i.tmp
done
```

```
# this munges them all into one big csv
for i in `cat $1`
    do
    grep -v port_a $i.csv >> $MUNGEFILE
    done

# put the column header back in
sed -i '1i port_a,port_b,total_packets_a2b,total_packets_b2a,unique_bytes_sent_a2b,unique_bytes_sent_b2a,session_duration,Class' $MUNGEFILE

**********end markeypcapmod script**********
```

****

*** Turning PCAPs into PDML (in order to give pdml2arff.py valid files on which to operate). ***

****

Base command: tshark – T pdml –r <infile> > <outfile>

If you have a directory full of PCAPSs to process you can create a file containing list of files in the directory then run the pcaps2pdml bash script (below) to batch process them.

**********start pcaps2pdml script**********

```
#!/bin/bash

# pcaps2pdml
# this script requires tshark to be installed.
# this takes one argument, it is a file which contains a list of files you want to operate on
# it will dump a pcap file in its entirety to pdml format (for my purposes which are to run them through pdml2arff.py)

# first, in the directory where the files which you wish to process are, do an "ls > <filelistname>" (i.e. pcapslist.malicious)
# and remove any lines that are not the files you're trying to operate on. (if there is a way to "ls > <file>" and NOT get the <file>
# in the file list someone please tell me.
# Then run: pcaps2pdml <filelist>
# this should process each file (pcap) in the <filelist> rendering a .pdml file corresponding with each pcap in the directory
# n.b. –due to potential corruption from whatever editor this may have been through – the "-" before the switches are short hyphens
```

```
for i in `cat $1`
        do
        # tshark does its magic and dumps the results to *.pdml files
        tshark –T pdml –r $i > $i.pdml
       pdml2arff.py $i.pdml > $i.arff
       done
```

**********end pcaps2pdml script**********

****

*** Turning PDMLs into ARFF. ***

****

Base command: pdml2arff.py <pdml.file.to.convert.to.arff>

If you have a directory full of PDMLs to process you can create a file containing list of files in the directory then run the pdmls2arff bash script (below) to batch process them.

**********start pdml2arff script**********

```
#!/bin/bash
```

```
# pdmls2arff
# this script requires python (2.7x) to be installed and expects pdml2arff.py to be in your path
# this takes one argument, it is a file which contains a list of files you want to operate on
# it will convert a pdml file in its entirety to ARFF  (Weka) format

# first, in the directory where the files which you wish to process are, do an "ls > <filelistname>" (i.e. pdml.list)
# and remove any lines that are not the files you are trying to operate on. (if there is a way to "ls > <file>" and NOT get the <file>
# in the file list someone please tell me.
# Then run: pdmls2arff <filelist>
# this should process each file (pdml) in the <filelist> rendering an ARFF file corresponding with each pdml in the directory
# n.b. –due to potential corruption from whatever editor this may have been through – the "-" before the switches are short hyphens

for i in `cat $1`
      do
      # tshark does its magic and dumps the results to *.pdml files
      tshark –T pdml –r $i > $i.pdml
     done

**********end pdmls2arff script**********
```

The script pdml2arff.py writes out the lines as it finds them in a pdml file, resulting in several lines for each packet_id. This does not seem to have much effect on the datamining algorithms themselves, but for human readability readability purposes it is convenient to have the multiple lines merged into one instance. We accomplish this using the merge.py script.

```
**********start merge.py script**********
#!/usr/bin/python2
# written by Tim Stello with input from Charlie Fowler 7/14

class PacketMerger:
    def __init__(self, files):
        for file in files:
            self.process_file(file)

    #
    # add value to a line at the correct pos if it is not blank or does not exist yet
    #
    def add_item_to_line(self, line, item, pos):
        if len(line) == pos:
            line.append(item.strip())
        else:
            if item in ['',' ','-']:
```

```
                return False
            line[pos] = item.strip()

    #
    # process a csv file and compact each line with the same first column value in to a single line
    #
    def process_file(self,file):
        lines = {}
        f = open(file,'r')

        for line in f:
            idx = 0
            key = ''
            for item in line.split(','):
                if idx is 0:
                    key = item
                    if item not in lines:
                        lines[key] = [key]
                else:
                    self.add_item_to_line(lines[key], item, idx)
                idx += 1
        self.print_output_lines(lines)
```

```
    def print_output_lines(self, lines):
        for key in sorted(lines):
            print(',').join(lines[key])


#
# Main
#
if __name__ == '__main__':
    import sys
    import re


    if len(sys.argv) < 2:
        print "merge.py <file1.csv> <file2.csv>..."
        sys.exit(0)
    else:
        sys.argv.pop(0) # remove 'merge.py' from argv
# send each file from command line into PacketMerger
PacketMerger(sys.argv)
**********end merge.py script**********




**********start pdml2arff.py script**********
```

```
#!/usr/bin/python
# written by Tim Stello with input from Charlie Fowler
# change the path (above) to reflect where you have python installed
#
# this script will take a tshark generated pdml file and turn it
# into an arff formatted file, suitable for ingestment by weka
# here is how to create the pdml file from pcap:
# tshark -T pdml -r <infile> > <outfile>
# (adding -V gets you no more data)
# usage of this script: pdml2arff.py <outfile> (outfile is pdml from above)
# ./pdml2arff.py <input_file> -o <output_file(optional)> -n (convert all strings to numerics

import csv

class myDialect(csv.Dialect):
    delimiter = ','
    quotechar = ""
    quoting = csv.QUOTE_NONNUMERIC
    lineterminator = "\n"
    doublequote = False
    skipinitialspace = False
```

```
#
# Define a simple class to wrap functions
#
class PdmlConvert:
    def __init__( self, templateString , numbers_only=False ):
        self.template = templateString
        self.numbers_only = numbers_only
        self.headers = []
        self.results = []
        self.packet_count = 1

    #
    # convert the given input to ARFF format
    #
    def convert_file( self, input_file , **kwargs ):
        fname,ext = self.parse_filename( input_file )
        output_file = kwargs.get( 'output_file', fname+'.arff' )
        self.parse_file( input_file )
        header = self.build_header( input_file )        # build the top section of output file
        self.write_to_file( header , output_file )      # write top section to output file
        self.append_array_of_dict_to_csv( output_file )     # write data to output file

    #
```

```
        #  uses xml.dom.minidom to parse input xml file
        #  - reads each packet -> proto -> field
        #  - creates a key/value results dict {} for each field
        #  - new fields are added to headers array
        #
        def parse_file( self , file ):
                from xml.dom import minidom          # load minidom
                self.clean_file( file )              # found a parsing error in input data, see clean_file
for info
                xmldoc = minidom.parse( file )       # use minidom to parse xml
                for packet in xmldoc.getElementsByTagName('packet'):     # for every packet ->
proto -> field...
                        self.parse_packet( packet )


        #
        #
        #
        def parse_packet( self , packet ):
                id = self.packet_count
                self.packet_count += 1
                for proto in packet.getElementsByTagName('proto'):
                        arf = self.create_arf( id )
                        arf = self.parse_proto_into_arf( arf , proto )
```

```
                    self.results.append( arf )
        #
        #
        #
        def parse_proto_into_arf( self , arf , proto ):
                proto_name = proto.getAttribute('name')
                for field in proto.getElementsByTagName('field'):
                        arf = self.parse_field_into_arf( proto_name , arf , field )
                return arf



        #
        # parse_field_into_arf ( proto_name , arf , field )
        #                 Adds any field or subfields to arf {} if it has a value
        #
        def parse_field_into_arf( self, proto_name , arf , field ):
                field_name = field.getAttribute('name')         # get name attribute of field
                name = self.build_name( field_name , proto_name ) #             build             name
grand.parent.child
                arf  =  self.append_key_value(  name  ,  self.get_value_from_field(  field  )  ,  arf  )
        # append key/val to arf dict {}

                # Some fields have children subfields with values
```

```
                for subfield in field.getElementsByTagName('field'):
                        sf_name = subfield.getAttribute('name')
                        name = self.build_name( sf_name , field.getAttribute('name') , proto_name
)
                        arf = self.append_key_value( name , self.get_value_from_field( subfield )
, arf )
                return arf


        #
        #
        #
        def append_key_value( self , key , value , map ):
                if value == '':
                        return map
                if not key in self.headers:
                        self.headers.append(key)
                map[key] = value
                return map


        #
        # Returns an unmaskedvalue or a vlue or '' from field attributes
        #
        def get_value_from_field( self , field ):
```

```
        if field.hasAttribute('unmaskedvalue'):
                return field.getAttribute('unmaskedvalue')
        elif field.hasAttribute('value'):
                return field.getAttribute('value')
        else:
                return ''


#
#
#
def build_name( self , name , parent , grand=''):
        ret = name
        if not str(name).startswith(parent):
                ret = parent + '.' + ret
        if not grand == '':
                if not ret.startswith(grand):
                        ret = grand + '.' + ret
        return ret


#
#
#
def create_arf( self , id ):
```

```
        if not 'packet_id' in self.headers:
                self.headers.append('packet_id')
        return { 'packet_id': id }


#
# This clean file is a simple xml cleaner of the <proto> </proto> element
# In the input files I've seen, there is an extra </proto> which shows up
# just before a '</packet>' in the data (often but not always).  So this function
# counts each opening '<proto' and closing '</proto>' and whenever we see an extra
# (count < 0) we do not output that extra one.  This seems to clean the file properly.
#
def clean_file( self , file ):
        import re
        stack = 0
        output = []
        for line in open( file , 'r'):
                if re.search('<proto',line):
                        stack += 1
                elif re.search('</proto>',line):
                        stack -= 1


                if stack >= 0:
                        output.append(line)
```

```
                else:
                        stack += 1


        o = open(file,'wb')
        for line in output:
                o.write( line )


    #
    # Appends and Array of Dictionaries to given filename
    # - inserts headers at beginning (of where appending happens)
    #
    def append_array_of_dict_to_csv( self , filename ):
        csvfile = open(filename, 'ab') # open file for appending
        dialect = myDialect()
        csvw = csv.DictWriter( csvfile , self.headers, '?' , dialect=dialect )   #      instantiate
DictWriter
        for kvs in self.results: # for every dict result, append dict to csv
                if self.numbers_only:
                        kvs = self.map2num( kvs )
                csvw.writerow( kvs )


    #
    # Writes text to filename
```

```
        #
        def write_to_file( self , text , filename ):
                f = open( filename , 'wb')
                f.write( text )


        #
        # Build header/top section of output file
        #
        def build_header( self , filename ):
                from string import Template
                text = Template( self.template )          # Template example:
                attr_str = ""      # temp = Template('this is a $INSERT')
                for attr in self.headers:            # print temp.substitute(INSERT='test')
    #                   attr_str += "@attribute " + attr + " STRING" + "\n" # use this if outputting
"string" data type
                        attr_str += "@attribute " + attr + " STRING" + "\n"  # use this if outputting
"numeric" data type
                return text.substitute(RELATION=filename,ATTRIBUTES=attr_str)


        #
        # Parse a filename into its base name and extension
        # returns [basename,ext] or 'Invalid Filename'
        #
```

```
def parse_filename( self , name ):
        import re
        r = re.search( r"(\S+)(\.\S{1,4})$", name )
        if r:
                return [ r.group(1) , r.group(2) ]
        else:
                raise 'Invalid Filename'


#
#  converts each value of the given map/dict to an integer using str2num
#
def map2num( self , m ):
        result = {}
        for k,v in m.iteritems():
                result[k] = self.str2num(v)
        return result


#
# Convert a string to a number (takes the ord value of each letter and
# combines it then converts it to int)
# i.e. str2num( 'abc' ); ord('a') = 97; "979899" => returns 979899 as int
#
def str2num( self , s ):
```

```
            if type(s) is int:
                return s
            num = ''
            for letter in s:
                o = ord(letter)
                num += str(o)
            return int(num)

    #
    #  Write errors to log
    #
    def error_log( self , message ):
            f = open('pdml.errors.log','wb')
            f.write( message )

# Template ARFF File
arff = '''
%
% This arff created by pdml2arff.py
% Written by Tim Stello with input from Charlie Fowler, spring 2013
% This script takes a pdml file created by tshark and converts it to arff
%
@relation $RELATION
```

```
    %
    %attributes
    %
    $ATTRIBUTES
    %
    @data
    %
    '''


    #
    # Main: this portion executes only when this file is executed
    # from the command line.  If you 'import' this file, this section
    # will not execute
    #
    if __name__ == '__main__':
        import sys
        usage = "./pdml2arffpy <input_file> -o <output_file (optional)> -n (convert all strings to numerics)\n"
        numbers_only = False
        if '-n' in sys.argv:
            numbers_only = True
            sys.argv.remove('-n')
        pdmlc = PdmlConvert(arff , numbers_only )
```

```
l = len(sys.argv)
if l == 2:
        pdmlc.convert_file( sys.argv[1] )
elif l == 4:
        pdmlc.convert_file( sys.argv[1] , { 'output_file':sys.argv[3] })
else:
        print usage
        sys.exit
```

**********end pdml2arff.py script**********

**********start preprocess.02.remove.attributes.sh script**********

```
!#/bin/bash

# remove attributes
# The same can be achieved with the following filter commandline (add -V to invert the selection):
# -V inverts matching sense of column indexes ($1)
# $1 is the list of indices $2 is the input file (to be changed) $3 is the output file (with attribs changed from numeric to nominal)

java weka.filters.unsupervised.attribute.Remove -R $1 -i $2 -o $3
```

**********end preprocess.02.remove.attributes.sh script**********

**********start preprocess.02.numeric.to.nominal.sh script**********

!#/bin/bash

change attributes datatype from numeric to nominal

valid values after -R are "1-7", "first-last"

-V inverts matching sense of column indexes ($1)

$1 is the list of indices $2 is the input file (to be changed) $3 is the output file (with attribs changed from numeric to nominal)

java weka.filters.unsupervised.attribute.NumericToNominal -R $1 -i $2 -o $3

**********end preprocess.02.numeric.to.nominal.sh script**********

**********start preprocess.do.merge.sh script**********

#!/bin/bash

8/2014 charlie fowler

This script takes an arff generated by pdml2arff.py splits out

the data section from the metadata, runs merge.py on the data

section then puts the file back together.

usage: do.merge.py.sh <infile>

```
# <infile> is the arff you wish to have merged down
# this script outputs a file called <infile>.merged.arff

# pull out arff metadata and put into a new file
preprocess.02a.get.metadata.from.arff.py $1 >> $1.merged.arff

# pull out arff data and put into a file
/bin/cat $1 | grep -v @ | grep -v % >> $1.onlydata

# run merge.py on the data file and add to new file
merge.py $1.onlydata >> $1.merged.arff

# cleanup
# delete the working file
/bin/rm $1.onlydata
**********end preprocess.do.merge.sh script**********

**********start stripqs.py script**********
#!/usr/bin/python2
# use this script to strip out lines in an arff file which are only unknown values

import re, sys
```

```
f = open(sys.argv[1], "r")


# put however many of these: '?', that there are that need to be gotten rid of

q = "'?','?'"


for line in f:

        if q not in line:

                print line.strip('\n')
```

**********end stripqs.py script**********

B. Appendix B - Run Results

This appendix contains the textual results from the various datamining runs accomplished using Weka. These are discussed in detail in chapter IV. Results

```
**************************************
 *** Classifier:J48 training (no scanning) data results ***
**************************************
=== Run information ===
Scheme:weka.classifiers.trees.J48 -C 0.25 -M 2
Relation:     traffic_analysis
Instances:    657
Attributes:   8
              port_a
              port_b
              total_packets_a2b
              total_packets_b2a
              unique_bytes_sent_a2b
              unique_bytes_sent_b2a
              session_duration
              Class
Test mode:10-fold cross-validation
=== Classifier model (full training set) ===
J48 pruned tree
```

---

port_a <= 1055

| port_b <= 447: MALICIOUS (38.0)

| port_b > 447: NORMAL (35.0)

port_a > 1055: NORMAL (584.0)

Number of Leaves : 3

Size of the tree : 5

Time taken to build model: 0.1 seconds

=== Stratified cross-validation ===

=== Summary ===

Correctly Classified Instances	657	100 %
Incorrectly Classified Instances	0	0 %
Kappa statistic	1	
Mean absolute error	0	
Root mean squared error	0	
Relative absolute error	0 %	
Root relative squared error	0 %	
Total Number of Instances	657	

=== Detailed Accuracy By Class ===

TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
1	0	1	1	1	1	NORMAL

1 0 1 1 1 1 MALICIOUS

Weighted Avg. 1 0 1 1 1 1

=== Confusion Matrix ===

a b <-- classified as

619 0 | a = NORMAL

0 38 | b = MALICIOUS

****

****** Classifier:J48 test (no scanning) data results ******

****

=== Run information ===

Scheme:weka.classifiers.trees.J48 -C 0.25 -M 2

Relation: traffic.analysis.training.no.scanning

Instances: 640

Attributes: 8

port_a

port_b

total_packets_a2b

total_packets_b2a

unique_bytes_sent_a2b

unique_bytes_sent_b2a

session_duration

Class

Test mode:10-fold cross-validation

=== Classifier model (full training set) ===

J48 pruned tree

----

port_a <= 1072

| port_b <= 1216: MALICIOUS (21.0)

| port_b > 1216: NORMAL (55.0)

port_a > 1072: NORMAL (564.0)

Number of Leaves : 3

Size of the tree : 5

Time taken to build model: 0.01 seconds

=== Stratified cross-validation ===

=== Summary ===

Correctly Classified Instances	637	99.5313 %
Incorrectly Classified Instances	3	0.4688 %
Kappa statistic	0.9207	
Mean absolute error	0.0048	
Root mean squared error	0.0684	
Relative absolute error	7.4575 %	
Root relative squared error	38.4047 %	
Total Number of Instances	640	

=== Detailed Accuracy By Class ===

TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class

1	0.143	0.995	1	0.998	0.946	NORMAL
0.857	0	1	0.857	0.923	0.946	MALICIOUS
Weighted Avg.	0.995	0.138	0.995	0.995	0.995	0.946

=== Confusion Matrix ===

a	b	<-- classified as
619	0	a = NORMAL
3	18	b = MALICIOUS

=== Re-evaluation on test set ===

User supplied test set

Relation: traffic.analysis.test.no.scanning.unlabeled

Instances: unknown (yet). Reading incrementally

Attributes: 8

=== Summary ===

Total Number of Instances	0
Ignored Class Unknown Instances	661

=== Detailed Accuracy By Class ===

TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
0	0	0	0	0	?	NORMAL
0	0	0	0	0	?	MALICIOUS
Weighted Avg. NaN	NaN	NaN	NaN	NaN	NaN	

=== Confusion Matrix ===

a b <-- classified as

0 0 | a = NORMAL

0 0 | b = MALICIOUS

=== Re-evaluation on test set ===

User supplied test set

Relation: traffic.analysis.test.no.scanning.unlabeled

Instances: unknown (yet). Reading incrementally

Attributes: 8

=== Predictions on test set ===

inst#, actual, predicted, error, probability distribution

1 ? 1:NORMAL + *1 0

2 ? 1:NORMAL + *1 0

3 ? 1:NORMAL + *1 0

----------truncated----------

618 ? 1:NORMAL + *1 0

619 ? 1:NORMAL + *1 0

620 ? 2:MALICIOU + 0 *1

621 ? 2:MALICIOU + 0 *1

622 ? 2:MALICIOU + 0 *1

----------truncated----------

659 ? 2:MALICIOU + 0 *1

660 ? 2:MALICIOU + 0 *1

661 ? 2:MALICIOU + 0 *1

=== Summary ===

Total Number of Instances 0

Ignored Class Unknown Instances 661

=== Detailed Accuracy By Class ===

TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
0	0	0	0	0	?	NORMAL
0	0	0	0	0	?	MALICIOUS
Weighted Avg. NaN	NaN	NaN	NaN	NaN	NaN	

=== Confusion Matrix ===

a b <-- classified as

0 0 | a = NORMAL

0 0 | b = MALICIOUS

****

****** Classifier:J48 labeled test data results (to double check algorithm's accuracy)******

****

=== Re-evaluation on test set ===

User supplied test set

Relation: traffic.analysis.test.no.scanning.labeled

Instances: unknown (yet). Reading incrementally

Attributes: 8

=== Predictions on test set ===

inst#, actual, predicted, error, probability distribution

1 1:NORMAL 1:NORMAL *1 0

2 1:NORMAL 1:NORMAL *1 0

3 1:NORMAL 1:NORMAL *1 0

----------truncated----------

617 1:NORMAL 1:NORMAL *1 0

618 1:NORMAL 1:NORMAL *1 0

619 1:NORMAL 1:NORMAL *1 0

620 2:MALICIOU 2:MALICIOU 0 *1

621 2:MALICIOU 2:MALICIOU 0 *1

622 2:MALICIOU 2:MALICIOU 0 *1

----------truncated----------

659 2:MALICIOU 2:MALICIOU 0 *1

660 2:MALICIOU 2:MALICIOU 0 *1

661 2:MALICIOU 2:MALICIOU 0 *1

=== Summary ===

Correctly Classified Instances	661	100 %
Incorrectly Classified Instances	0	0 %
Kappa statistic	1	
Mean absolute error	0	
Root mean squared error	0	
Total Number of Instances	661	

=== Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
	1	0	1	1	1	1	NORMAL
	1	0	1	1	1	1	MALICIOUS
Weighted Avg.	1	0	1	1	1	1	

=== Confusion Matrix ===

```
  a   b   <-- classified as
 619   0 |   a = NORMAL
   0  42 |   b = MALICIOUS
```

****

****

***** Classifier:J48 labeled test data results with a port scan through the J48 classifier *****

****

****

```
=== Run information ===
Scheme:weka.classifiers.trees.J48 -C 0.25 -M 2
Relation:     traffic.analysis.training
Instances:    3636
Attributes:   8
              port_a
              port_b
              total_packets_a2b
              total_packets_b2a
              unique_bytes_sent_a2b
              unique_bytes_sent_b2a
              session_duration
              Class
Test mode:10-fold cross-validation
```

=== Classifier model (full training set) ===

J48 pruned tree

----

port_a <= 45391

| port_a <= 1072

| | port_b <= 1233: MALICIOUS (21.0)

| | port_b > 1233: NORMAL (55.0)

| port_a > 1072: NORMAL (538.0)

port_a > 45391

| port_a <= 47584: SCANNING (2996.0)

| port_a > 47584: NORMAL (26.0)

Number of Leaves : 5

Size of the tree : 9

Time taken to build model: 0.35 seconds

=== Stratified cross-validation ===

=== Summary ===

Correctly Classified Instances	3630	99.835 %
Incorrectly Classified Instances	6	0.165 %
Kappa statistic	0.9943	
Mean absolute error	0.0011	
Root mean squared error	0.0329	
Relative absolute error	0.5782 %	
Root relative squared error	10.5569 %	
Total Number of Instances	3636	

=== Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
	0.997	0.001	0.994	0.997	0.995	0.998	NORMAL
	0.857	0	1	0.857	0.923	0.975	MALICIOUS
	1	0.003	0.999	1	0.999	0.998	SCANNING
Weighted Avg.	0.998	0.003	0.998	0.998	0.998	0.998	

=== Confusion Matrix ===

a	b	c	<-- classified as
617	0	2	a = NORMAL

3 18 0 | b = MALICIOUS

1 0 2995 | c = SCANNING

=== Re-evaluation on test set ===

User supplied test set

Relation: traffic.analysis.test

Instances: unknown (yet). Reading incrementally

Attributes: 8

=== Summary ===

Total Number of Instances 0

Ignored Class Unknown Instances 2642

=== Detailed Accuracy By Class ===

TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
0	0	0	0	0	?	NORMAL
0	0	0	0	0	?	MALICIOUS
0	0	0	0	0	?	SCANNING
Weighted Avg. NaN	NaN	NaN	NaN	NaN	NaN	

=== Confusion Matrix ===

a b c <-- classified as

0 0 0 | a = NORMAL

0 0 0 | b = MALICIOUS

0 0 0 | c = SCANNING

=== Re-evaluation on test set ===

User supplied test set

Relation: traffic.analysis.test

Instances: unknown (yet). Reading incrementally

Attributes: 8

=== Predictions on test set ===

inst#, actual, predicted, error, probability distribution

1 ? 1:NORMAL + *1 0 0

2 ? 1:NORMAL + *1 0 0

3 ? 1:NORMAL + *1 0 0

----------truncated----------

617	?	1:NORMAL	+	*1	0	0
618	?	1:NORMAL	+	*1	0	0
619	?	1:NORMAL	+	*1	0	0
620	?	2:MALICIOU	+	0	*1	0
621	?	2:MALICIOU	+	0	*1	0
622	?	2:MALICIOU	+	0	*1	0

----------truncated----------

638	?	2:MALICIOU	+	0	*1	0
639	?	2:MALICIOU	+	0	*1	0
640	?	2:MALICIOU	+	0	*1	0
641	?	1:NORMAL	+	*1	0	0
642	?	1:NORMAL	+	*1	0	0
643	?	1:NORMAL	+	*1	0	0

----------truncated----------

2639	?	1:NORMAL	+	*1	0	0
2640	?	1:NORMAL	+	*1	0	0
2641	?	1:NORMAL	+	*1	0	0
2642	?	1:NORMAL	+	*1	0	0

=== Summary ===

Total Number of Instances	0
Ignored Class Unknown Instances	2642

=== Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
	0	0	0	0	0	?	NORMAL
	0	0	0	0	0	?	MALICIOUS
	0	0	0	0	0	?	SCANNING
Weighted Avg.	NaN	NaN	NaN	NaN	NaN	NaN	

=== Confusion Matrix ===

```
 a b c   <-- classified as
 0 0 0 | a = NORMAL
 0 0 0 | b = MALICIOUS
 0 0 0 | c = SCANNING
```

=== Re-evaluation on test set ===

User supplied test set

Relation: traffic.analysis.test.labeled

Instances: unknown (yet). Reading incrementally

Attributes: 8

=== Predictions on test set ===

inst#, actual, predicted, error, probability distribution

1 1:NORMAL 1:NORMAL *1 0 0

2 1:NORMAL 1:NORMAL *1 0 0

3 1:NORMAL 1:NORMAL *1 0 0

----------truncated----------

617 1:NORMAL 1:NORMAL *1 0 0

618 1:NORMAL 1:NORMAL *1 0 0

619 1:NORMAL 1:NORMAL *1 0 0

620 2:MALICIOU 2:MALICIOU 0 *1 0

621 2:MALICIOU 2:MALICIOU 0 *1 0

622 2:MALICIOU 2:MALICIOU 0 *1 0

----------truncated----------

638 2:MALICIOU 2:MALICIOU 0 *1 0

639 2:MALICIOU 2:MALICIOU 0 *1 0

640 2:MALICIOU 2:MALICIOU 0 *1 0

641 3:SCANNING 1:NORMAL + *1 0 0

642 3:SCANNING 1:NORMAL + *1 0 0

643 3:SCANNING 1:NORMAL + *1 0 0

----------truncated----------

2640 3:SCANNING 1:NORMAL + *1 0 0

2641 3:SCANNING 1:NORMAL + *1 0 0

2642 3:SCANNING 1:NORMAL + *1 0 0

=== Summary ===

Correctly Classified Instances	640	24.2241 %
Incorrectly Classified Instances	2002	75.7759 %
Kappa statistic	0.0127	
Mean absolute error	0.5052	
Root mean squared error	0.7108	
Total Number of Instances	2642	

=== Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
	1	0.99	0.236	1	0.382	0.505	NORMAL
	1	0	1	1	1	1	MALICIOUS
	0	0	0	0	0	0.5	SCANNING
Weighted Avg.	0.242	0.232	0.063	0.242	0.097	0.505	

=== Confusion Matrix ===

a	b	c	<-- classified as

619 0 0 | a = NORMAL

0 21 0 | b = MALICIOUS

2002 0 0 | c = SCANNING

****

****** Classifier:J48 training data results with a port scan ******

****

****

=== Run information ===

Scheme:weka.classifiers.trees.J48 -C 0.25 -M 2

Relation: traffic.analysis.training

Instances: 3636

Attributes: 8

port_a

port_b

total_packets_a2b

total_packets_b2a

unique_bytes_sent_a2b

unique_bytes_sent_b2a

session_duration

Class

Test mode:10-fold cross-validation

=== Classifier model (full training set) ===

J48 pruned tree

----

port_a <= 45391

| port_a <= 1072

| | port_b <= 1233: MALICIOUS (21.0)

| | port_b > 1233: NORMAL (55.0)

| port_a > 1072: NORMAL (538.0)

port_a > 45391

| port_a <= 47584: SCANNING (2996.0)

| port_a > 47584: NORMAL (26.0)

Number of Leaves : 5

Size of the tree : 9

Time taken to build model: 0.35 seconds

=== Stratified cross-validation ===

=== Summary ===

Correctly Classified Instances	3630	99.835 %
Incorrectly Classified Instances	6	0.165 %
Kappa statistic	0.9943	
Mean absolute error	0.0011	
Root mean squared error	0.0329	
Relative absolute error	0.5782 %	
Root relative squared error	10.5569 %	
Total Number of Instances	3636	

=== Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
	0.997	0.001	0.994	0.997	0.995	0.998	NORMAL
	0.857	0	1	0.857	0.923	0.975	MALICIOUS
	1	0.003	0.999	1	0.999	0.998	SCANNING
Weighted Avg.	0.998	0.003	0.998	0.998	0.998	0.998	

=== Confusion Matrix ===

```
   a    b    c   <-- classified as
 617    0    2 |    a = NORMAL
   3   18    0 |    b = MALICIOUS
   1    0 2995 |    c = SCANNING
```

****

***** Classifier:J48 test data results with a port scan ***

****

=== Run information ===

Scheme:weka.classifiers.trees.J48 -C 0.25 -M 2

Relation: traffic.analysis.training

Instances: 3636

Attributes: 8

port_a

port_b

total_packets_a2b

total_packets_b2a

unique_bytes_sent_a2b

unique_bytes_sent_b2a

session_duration

Class

Test mode:10-fold cross-validation

=== Classifier model (full training set) ===

J48 pruned tree

---

port_a <= 45391

| port_a <= 1072

| | port_b <= 1233: MALICIOUS (21.0)

| | port_b > 1233: NORMAL (55.0)

| port_a > 1072: NORMAL (538.0)

port_a > 45391

| port_a <= 47584: SCANNING (2996.0)

| port_a > 47584: NORMAL (26.0)

Number of Leaves : 5

Size of the tree : 9

Time taken to build model: 0.35 seconds

=== Stratified cross-validation ===

=== Summary ===

Correctly Classified Instances	3630	99.835 %
Incorrectly Classified Instances	6	0.165 %
Kappa statistic	0.9943	
Mean absolute error	0.0011	
Root mean squared error	0.0329	
Relative absolute error	0.5782 %	
Root relative squared error	10.5569 %	
Total Number of Instances	3636	

=== Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
	0.997	0.001	0.994	0.997	0.995	0.998	NORMAL
	0.857	0	1	0.857	0.923	0.975	MALICIOUS
	1	0.003	0.999	1	0.999	0.998	SCANNING
Weighted Avg.	0.998	0.003	0.998	0.998	0.998	0.998	

=== Confusion Matrix ===

a	b	c	<-- classified as
617	0	2	a = NORMAL
3	18	0	b = MALICIOUS
1	0	2995	c = SCANNING

=== Re-evaluation on test set ===

User supplied test set

Relation: traffic.analysis.test

Instances: unknown (yet). Reading incrementally

Attributes: 8

=== Summary ===

Total Number of Instances 0

Ignored Class Unknown Instances 2642

=== Detailed Accuracy By Class ===

TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
0	0	0	0	0	?	NORMAL
0	0	0	0	0	?	MALICIOUS
0	0	0	0	0	?	SCANNING
Weighted Avg. NaN	NaN	NaN	NaN	NaN	NaN	

=== Confusion Matrix ===

a b c <-- classified as

0 0 0 | a = NORMAL

0 0 0 | b = MALICIOUS

0 0 0 | c = SCANNING

=== Re-evaluation on test set ===

User supplied test set

Relation: traffic.analysis.test

Instances: unknown (yet). Reading incrementally

Attributes: 8

=== Predictions on test set ===

inst#, actual, predicted, error, probability distribution

1 ? 1:NORMAL + *1 0 0

2 ? 1:NORMAL + *1 0 0

3 ? 1:NORMAL + *1 0 0

----------truncated----------

617 ? 1:NORMAL + *1 0 0

618 ? 1:NORMAL + *1 0 0

619 ? 1:NORMAL + *1 0 0

620 ? 2:MALICIOU + 0 *1 0

621 ? 2:MALICIOU + 0 *1 0

622 ? 2:MALICIOU + 0 *1 0

----------truncated----------

638 ? 2:MALICIOU + 0 *1 0

639 ? 2:MALICIOU + 0 *1 0

640 ? 2:MALICIOU + 0 *1 0

641 ? 1:NORMAL + *1 0 0

642 ? 1:NORMAL + *1 0 0

643 ? 1:NORMAL + *1 0 0

----------truncated----------

2640 ? 1:NORMAL + *1 0 0

2641 ? 1:NORMAL + *1 0 0

2642 ? 1:NORMAL + *1 0 0

=== Summary ===

Total Number of Instances 0

Ignored Class Unknown Instances 2642

=== Detailed Accuracy By Class ===

TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
0	0	0	0	0	?	NORMAL
0	0	0	0	0	?	MALICIOUS
0	0	0	0	0	?	SCANNING
Weighted Avg.	NaN	NaN	NaN	NaN	NaN	NaN

=== Confusion Matrix ===

a b c <-- classified as

0 0 0 | a = NORMAL

0 0 0 | b = MALICIOUS

0 0 0 | c = SCANNING

****

****

**** Clusterer:SimpleKMeans nmap.scan.37 training results ****

****

=== Run information ===

Scheme:weka.clusterers.SimpleKMeans -N 2 -A "weka.core.EuclideanDistance -R first-last" -I 500 -S 10

Relation: nmap.scan.37.pcap.pdml-weka.filters.unsupervised.attribute.Remove-R2-34,36-38,40-90-weka.filters.unsupervised.attribute.StringToNominal-R2,3

Instances: 2013

Attributes: 3

packet_id

ip.src

ip.dst

Test mode:evaluate on training data

=== Model and evaluation on training set ===

kMeans

======

Number of iterations: 9

Within cluster sum of squared errors: 53.979187375745546

Missing values globally replaced with mean/mode

Cluster centroids:

Cluster#

Attribute Full Data 0 1

(2013) (1006) (1007)

==========

packet_id 1007 503.5 1510

ip.src 0a00020f 0a00020f 0a00020f

ip.dst c0a80125 c0a80125 c0a80125

Time taken to build model (full training data) : 0.11 seconds

=== Model and evaluation on training set ===

Clustered Instances

0 1006 (50%)

1 1007 (50%)

****

***** *** Clusterer:SimpleKMeans nmap.scan.37.&.example.com-5 training results ***

****

***** === Run information ===

Scheme:weka.clusterers.SimpleKMeans -N 2 -A "weka.core.EuclideanDistance -R first-last" -I 500 -S 10

Relation: nmap.scan.37.&.example.com-5

Instances: 2013

Attributes: 2

ip.src

ip.dst

Test mode:evaluate on training data

=== Model and evaluation on training set ===

kMeans

======

Number of iterations: 2

Within cluster sum of squared errors: 0.0

Missing values globally replaced with mean/mode

Cluster centroids:

Cluster#

Attribute Full Data 0 1

(2013) (2007) (6)

==========

ip.src 0a00020f 0a00020f c0a80125

ip.dst c0a80125 c0a80125 0a00020f

Time taken to build model (full training data) : 0.03 seconds

=== Model and evaluation on training set ===

Clustered Instances

0 2007 (100%)

1 6 (0%)

****

*** Clusterer:Cobweb nmap.scan.37.&.example.com-5 training results ***

****

=== Run information ===

Scheme:weka.clusterers.Cobweb -A 1.0 -C 0.0028209479177387815 -S 42

Relation: nmap.scan.37.&.example.com-5

Instances: 2013

Attributes: 2

ip.src

ip.dst

Test mode:evaluate on training data

=== Model and evaluation on training set ===

Number of merges: 0

Number of splits: 0

Number of clusters: 3

node 0 [2013]

| leaf 1 [2007]

node 0 [2013]

| leaf 2 [6]

Time taken to build model (full training data) : 0.71 seconds

=== Model and evaluation on training set ===

Clustered Instances

1 2007 (100%)

2 6 (0%)

****

*** *Clusterer:Cobweb example.com-5 training results* ***

****

=== Run information ===

Scheme:weka.clusterers.Cobweb -A 2.0 -C 0.0028209479177387815 -S 42

Relation: example.com-5.pcap.pdml-weka.filters.unsupervised.attribute.Remove-R2-35,37-39,41-2322-weka.filters.unsupervised.attribute.Remove-R1-weka.filters.unsupervised.attribute.StringToNominal-R1-2

Instances: 1183

Attributes: 2

ip.src

ip.dst

Test mode:evaluate on training data

=== Model and evaluation on training set ===

Number of merges: 324

Number of splits: 275

Number of clusters: 410

--------snip-------

| | | | | leaf 407 [1]

| | | node 283 [121]

| | | | leaf 408 [7]

| | node 281 [184]

| | | leaf 409 [22]

Time taken to build model (full training data) : 50.61 seconds

=== Model and evaluation on training set ===

Clustered Instances

3 11 (1%)

4 2 (0%)

--------snip----------

****

*** Clusterer:SimpleKMeans 2 clusters example.com-5 training results ***

****

=== Run information ===

Scheme:weka.clusterers.SimpleKMeans -N 2 -A "weka.core.EuclideanDistance -R first-last" -I 500 -S 10

Relation: example.com-5.pcap.pdml-weka.filters.unsupervised.attribute.Remove-R2-35,37-39,41-2322-weka.filters.unsupervised.attribute.Remove-R1-weka.filters.unsupervised.attribute.StringToNominal-R1-2

Instances: 1183

Attributes: 2

ip.src

ip.dst

Test mode:evaluate on training data

=== Model and evaluation on training set ===

kMeans

======

Number of iterations: 3

Within cluster sum of squared errors: 1771.0

Missing values globally replaced with mean/mode

Cluster centroids:

Cluster#

Attribute Full Data 0 1

(1183) (1055) (128)

==========

ip.src c0a80a80 c0a80a80 c0a80a65

ip.dst c0a80a80 c0a80a80 c0a80a7d

Time taken to build model (full training data) : 0.02 seconds

=== Model and evaluation on training set ===

Clustered Instances

0 1055 (89%)

1 128 (11%)

****

***** *Clusterer:SimpleKMeans 10 clusters example.com-5 training results* *****

****

=== Run information ===

Scheme:weka.clusterers.SimpleKMeans -N 10 -A "weka.core.EuclideanDistance -R first-last" -I 500 -S 10

Relation: example.com-5.pcap.pdml-weka.filters.unsupervised.attribute.Remove-R2-35,37-39,41-2322-weka.filters.unsupervised.attribute.Remove-R1-weka.filters.unsupervised.attribute.StringToNominal-R1-2

Instances: 1183

Attributes: 2

ip.src

ip.dst

Test mode:evaluate on training data

=== Model and evaluation on training set ===

kMeans

======

Number of iterations: 3

Within cluster sum of squared errors: 913.0

Missing values globally replaced with mean/mode

Cluster centroids:

Cluster#

Attribute Full Data 0 1 2 3 4 5 6 7 8 9

(1183) (386) (110) (131) (70) (266) (66) (90) (45) (18) (1)

==========

ip.src c0a80a80 c0a80a64 c0a80a65 c0a80a78 c0a80a67 c0a80a80 c0a80a64 c0a80a80 c0a80a7d c0a80a7e 4d330618

ip.dst c0a80a80 c0a80a80 c0a80a7d c0a80a66 52604004 c0a80a64 effffffa c0a80a65 c0a80a65 c0a80a65 c0a80a80

Time taken to build model (full training data) : 0.02 seconds

=== Model and evaluation on training set ===

Clustered Instances

0 386 (33%)

1 110 (9%)

-------snip---------

****

*** *Clusterer:SimpleKMeans 10 clusters nmap.scan.37.&.example.com-5 training results* ***

****

=== Run information ===

Scheme:weka.clusterers.SimpleKMeans -N 10 -A "weka.core.EuclideanDistance -R first-last" -I 500 -S 10

Relation: nmap.scan.37.&.example.com-5

Instances: 2013

Attributes: 2

ip.src

ip.dst

Test mode:evaluate on training data

=== Model and evaluation on training set ===

kMeans

======

Number of iterations: 2

Within cluster sum of squared errors: 0.0

Missing values globally replaced with mean/mode

Cluster centroids:

		Cluster#	
Attribute	Full Data	0	1
	(2013)	(2007)	(6)
ip.src	0a00020f	0a00020f	c0a80125
ip.dst	c0a80125	c0a80125	0a00020f

Time taken to build model (full training data) : 0.06 seconds

=== Model and evaluation on training set ===

Clustered Instances

0 2007 (100%)

1 6 (0%)

x: instance_number y: ip.src -> interesting

****

****

***** *Clusterer:Cobweb traffic.analysis.test.no.scanning training results* *****

****

== Run information ===

Scheme:weka.clusterers.Cobweb -A 1.0 -C 0.0028209479177387815 -S 42

Relation: traffic.analysis.test.noscanning

Instances: 640

Attributes: 8

port_a

port_b

total_packets_a2b

total_packets_b2a

unique_bytes_sent_a2b

unique_bytes_sent_b2a

session_duration

Class

Test mode:evaluate on training data

=== Model and evaluation on training set ===

Number of merges: 243

Number of splits: 204

Number of clusters: 760

node 0 [640]

| node 1 [212]

| | node 2 [2]

--------snip----------

Time taken to build model (full training data) : 13.28 seconds

=== Model and evaluation on training set ===

Clustered Instances

1 6 (1%)

3 1 (0%)

4 1 (0%)

--------snip----------

****

****** Associator:Apriori zeus-sample-2.pcap (anomaly) results ******

****

=== Run information ===

Scheme: weka.associations.Apriori -N 10 -T 0 -C 0.9 -D 0.05 -U 1.0 -M 0.1 -S -1.0 -c -1

Relation: ./output/zeus-sample-2.pcap.pdml-weka.filters.unsupervised.attribute.Remove-R1-34,36-38,40-41,44-47,49-53,59-110-weka.filters.unsupervised.attribute.StringToNominal-Rfirst-last

Instances: 567

Attributes: 10

ip.src

ip.dst

tcp.srcport

tcp.dstport

tcp.flags

tcp.flags.ack

tcp.flags.push

tcp.flags.reset

tcp.flags.syn

tcp.flags.fin

=== Associator model (full training set) ===

Apriori

=======

Minimum support: 0.8 (454 instances)

Minimum metric <confidence>: 0.9

Number of cycles performed: 4

Generated sets of large itemsets:

Size of set of large itemsets L(1): 10

Size of set of large itemsets L(2): 29

Size of set of large itemsets L(3): 56

Size of set of large itemsets L(4): 70

Size of set of large itemsets L(5): 56

Size of set of large itemsets L(6): 28

Size of set of large itemsets L(7): 8

Size of set of large itemsets L(8): 1

Best rules found:

1. ip.dst=? 457 ==> ip.src=? 457 conf:(1)
2. ip.src=? 457 ==> ip.dst=? 457 conf:(1)
3. tcp.dstport=? 457 ==> tcp.srcport=? 457 conf:(1)
4. tcp.srcport=? 457 ==> tcp.dstport=? 457 conf:(1)
5. tcp.flags=? 457 ==> tcp.srcport=? 457 conf:(1)
6. tcp.srcport=? 457 ==> tcp.flags=? 457 conf:(1)
7. tcp.flags.ack=? 457 ==> tcp.srcport=? 457 conf:(1)
8. tcp.srcport=? 457 ==> tcp.flags.ack=? 457 conf:(1)
9. tcp.flags.push=? 457 ==> tcp.srcport=? 457 conf:(1)
10. tcp.srcport=? 457 ==> tcp.flags.push=? 457 conf:(1)

C. Appendix C – Glossary

This is a limited glossary containing definitions for words used in intrusion detection and multi-agent systems disciplines

Hybrid Intelligent System - "The integration of different learning and adaptation techniques to overcome individual limitations and to achieve synergetic effects through the hybridization or fusion of these techniques" [11]

Or

"Any system that uses at least one intelligence paradigm in combination with another…" [137]

MapReduce

MapReduce is a programming model and an associated implementation for processing and generating large data sets. Users specify a map function that processes a key/value pair to generate a set of intermediate key/value pairs, and a reduce function that merges all intermediate values associated with the same intermediate key. Many real world tasks are expressible in this model, as shown in the paper.

Programs written in this functional style are automatically parallelized and executed on a large cluster of commodity machines. The run-time system takes care of the details of partitioning the input data, scheduling the program's execution across a set of machines, handling machine failures, and managing the required inter-machine communication. This allows programmers without any experience with parallel and distributed systems to easily utilize the resources of a large distributed system. [76]

Multi-Agent Systems

- "A multiagent system uses groups of self-directed agents working together to achieve a common goal." [60]

NP-Complete

The *complexity class* of *decision problems* for which answers can be checked for correctness, given a *certificate*, by an algorithm whose run time is *polynomial* in the size of the input (that is, it is *NP*) and no other NP problem is more than a polynomial factor harder. Informally, a problem is NP-complete if answers can be verified quickly, and a quick algorithm to solve this problem can be used to solve all other NP problems quickly. [138]

PWN

A leetspeak slang term derived from the verb "own", as meaning to appropriate or to conquer to gain ownership. The term implies domination or humiliation of a rival, used primarily in the Internet gaming culture to taunt an opponent who has just been soundly defeated (e.g., "You just got pwned!").

In hacker jargon, pwn means to compromise or control, specifically another computer (server or PC), web site, gateway device, or application. It is synonymous with one of the definitions of hacking or cracking. The Pwnie Awards are awarded by a group of security researchers.

Popularity of the term among teenagers rose in the mid-2000s, with the spread from the Internet written form to use in spoken language. [139]

Small World

"Small world networks are those in which most nodes are highly connected to their neighbours, that is, the nodes are highly clustered and have shorter paths between them." [140]

> A type of mathematical graph in which most nodes are not neighbors of one another, but most nodes can be reached from every other by a small number of hops or steps. A small world network, where nodes represent people and edges connect people that know each other, captures the small world phenomenon of strangers being linked by a mutual acquaintance. [141]

Stigmergy

> Stigmergy is a mechanism of spontaneous, indirect coordination between agents or actions, where the trace left in the environment by an action stimulates the performance of a subsequent action, by the same or a different agent. Stigmergy is a form of self-organization. It produces complex, apparently intelligent structures, without need for any planning, control, or even communication between the agents. As such it supports efficient collaboration between extremely simple agents, who lack any memory, intelligence or even awareness of each other. [142]

Swarm Intelligence

> Swarm Intelligence (SI) is the property of a system whereby the collective behaviors of (unsophisticated) agents interacting locally with their environment cause coherent functional global patterns to emerge. [143]

D. Appendix D – Acronyms

This is a limited list containing acronyms used in intrusion detection and multi-agent systems disciplines

ABAN - Agent-Based Active Network

ACO - Ant Colony Optimization

ACPL - Agent Communication Programming Language

ADMF - Adaptive Decision Making Framework

BDI - Belief, Desire, Intention, Goals within BDI: query, perform, achieve, maintain, cease, avoid, optimize, test, preserve. [68]

CAS - Complex Adaptive systems

CNET - Contract Net

CoABS - Control of Agent Based Systems

CVRP - Capacitated Vehicle Routing Problem

DAML - DARPA Agent Markup Language

DynCNET - Dynamic Contract Net

FIPA

Foundation for Intelligent Physical Agents: FIPA is an IEEE Computer Society standards organization that promotes agent-based technology and the interoperability of its standards with other technologies. [110]

FIRE - "FIRE is from “fides” (Latin for “trust”) and “reputation”. In the Ramayana legend of India, Sita proved the purity of her character by passing through the raging flames." [144]

FiTA - Field Task Assignment

FlexFeed - Flexible Data Feeds Framework

GA - Genetic Algorithm

GVR - Genetic Vehicle Representation

KQML - Knowledge Query and Manipulation Language

MANET - Mobile ad-hoc network

MAS - Multi-Agent Systems

MBM - Multi-agent Based Models

PSO - Particle Swarm Optimization

RDBMS - Relational Database Management System

SANET - Sensor/Actuator Networks

TCM - Traditional Chinese Medicine

VRP - Vehicle Routing Problem

WSN - Wireless Sensor Networks

E. Appendix E – Software List

Agent-0 - A multi agent systems - agents with mental state

Desire - "framework for DEsign and Specification of Interacting REasoning components" … "explicitly models the knowledge, interaction, and coordination of complex tasks and reasoning capabilities in agent systems." [145]

Hadoop!

> **Apache Hadoop** is a Java software framework that supports data-intensive distributed applications under a free license. It enables applications to work with thousands of nodes and petabytes of data. Hadoop was inspired by Google's MapReduce and Google File System (GFS) papers. [146] [81]

Honeywall –

> Honeywall CDROM is our primary high-interaction tool for capturing, controling and analyzing attacks. It creates an architecture that allows you to deploy both low-interaction and high-interaction honeypots, but is designed primarily for high-interaction. [120]

Repast - agent based modeling toolkit [147]

SeeCoast - is software that

> …extends the US Coast Guard Port Security and Monitoring system by adding capabilities to detect, classify, and track vessels using electro-optic and infrared cameras, and also uses learned normalcy models of vessel activities in order to generate alert cues for the watch-standers when anomalous behaviors occur. SeeCoast fuses the video data with radar detections and Automatic Identification System (AIS) transponder data in order to generate composite fused tracks for

vessels approaching the port, as well as for vessels already in the port. Then, SeeCoast applies rule-based and learning-based pattern recognition algorithms to alert the watch-standers to unsafe, illegal, threatening, and other anomalous vessel activities. [148]

VIPAR

Open Source Intelligence Analysis - intelligent software agents have been successfully developed to address challenges facing the military and intelligence community in quickly gathering and organizing massive amounts of information then distill that information into a form directly and explicitly amenable for use by an analyst. [149]

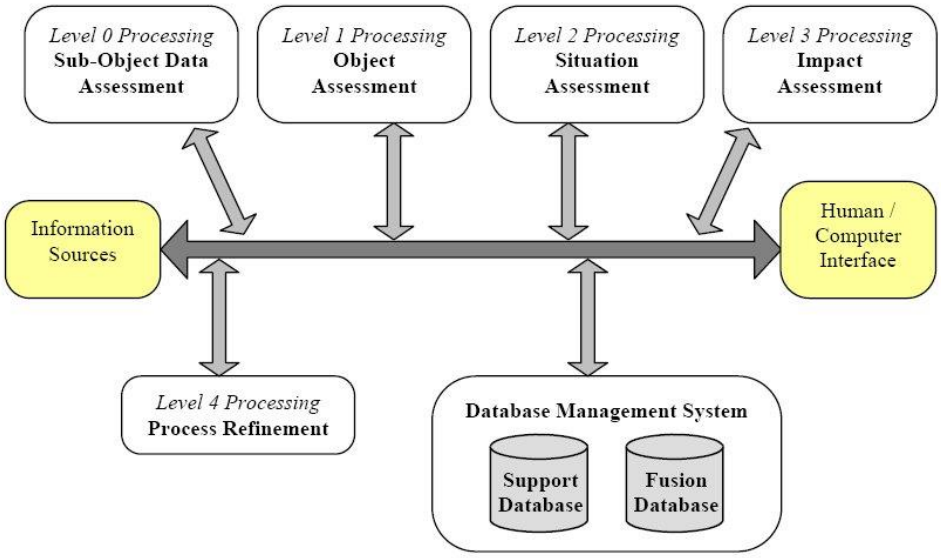

Figure 37. VIPAR Information Fusion Process Model [149]

JADE - used by TCMMADM [5]

JADE (Java Agent DEvelopment Framework) is a software framework fully implemented in Java language. It simplifies the implementation of multi-agent systems through a middle-ware that claims to comply with the FIPA specifications

and through a set of tools that supports the debugging and deployment phase. The agent platform can be distributed across machines (which not even need to share the same OS) and the configuration can be controlled via a remote GUI. The configuration can be even changed at run-time by moving agents from one machine to another one, as and when required. The only system requirement is the Java Run Time version 1.4 or later.

The communication architecture offers flexible and efficient messaging, where JADE creates and manages a queue of incoming ACL messages, private to each agent; agents can access their queue via a combination of several modes: blocking, polling, timeout and pattern matching based. The full FIPA communication model has been implemented and its components have been made clearly distinct and fully integrated: interaction protocols, envelope, ACL, content languages, encoding schemes, ontologies and, finally, transport protocols. The transport mechanism, in particular, is like a chameleon because it adapts to each situation, by transparently choosing the best available protocol. Java RMI, event-notification, HTTP and IIOP are currently used, but more protocols can be easily added. Most of the interaction protocols defined by FIPA are already available and can be instantiated after defining the application-dependent behaviour of each state of the protocol. SL and agent management ontology have been implemented already, as well as the support for user-defined content languages and ontologies that can be implemented, registered with agents, and automatically used by the framework. JADE has also been integrated with JESS, a Java shell of CLIPS, in order to exploit its reasoning capabilities. JADE is being used by a number of

companies and academic groups, both members and non-members of FIPA, such as BT, CNET, NHK, Imperial College, IRST, KPN, University of Helsinky, INRIA, ATOS and many others. It has been recently made available under Open Source License. [150]

Jason - "Jason is a fully-fledged interpreter for an extended version of AgentSpeak, a BDI agent-oriented logic programming language, and is implemented in Java. Using SACI, a multi-agent system can be distributed over a network effortlessly." [151]

Nuin - "A Java framework for building belief-desire-intention agents, with a particular emphasis on Semantic Web agents." [152]

SPARK

SPARK (SRI Procedural Agent Realization Kit) is a new agent framework, under development at the Artificial Intelligence Center of SRI International. SPARK builds on the success of its predecessor, PRS, and shares the same Belief Desire Intention (BDI) model of rationality. SPARK has been developed to support the construction of practical agent systems, and contains sophisticated mechanisms for encoding and controlling agent behavior. At the same time, SPARK has a well-defined semantic model that is intended to support reasoning about the agents' knowledge and execution. [152]

These are just a few of the most prominent BDI platforms, others are: The Living Systems Technology Suite, Tryllian ADK, Cougaar, CybelePro, Soar, Agent Factory, Jackdaw, Jam, Agent Business Force, Emorphia FIPA OS, Madkit, Grasshopper, April Agent Platform, DIET Agents, JIAC, Lost Wax, Sage, Semoa, A-Globe, AMETAS, INDUS, ABLE, Aglets, ACT-R, Micro-PSI, Retsina, Practionist, Open PRS, AgentScape, CHAP and Lisa. [59]

F. Appendix F - Software installation instructions

1) Installing Weka

Installing Weka is straightforward on Ubuntu using the following command:

user@host ~/ $ sudo apt-get install weka

Installing the software on a Windows based host is fairly straightforward as well, simply download the setup file from their website [29] and follow the instructions. Weka's site is also replete with help files, tutorials and a responsive forum.

2) Installing SPADE

The steps to install SPADE are also fairly straightforward; simply running the following command will install the SPADE framework on a Linux based host with Python 2.7.x and Python-pip already installed:

user@host ~/ $ sudo pip install spade

Although the SPADE installation instructions call for a fully qualified domain name, the host does not need to be installed on an Internet reachable host with a FQDN as usually understood. Defining a name in /etc/hosts or even passing the ip address will suffice as long as the "naming convention" is consistent throughout all configurations. Having said that, the platform *does* seem to require that the "FQDN" have at least one "." in it, for instance, "host.example" rather than just "host."

Having installed SPADE, in order to configure it, run the command below which takes one argument, which is the "fully qualified hostname" (discussed above) of the host where the SPADE framework has been set up:

user@host ~/ $ sudo configure.py <fully qualified domain name>

Then run the following command which will launch the framework:

user@host ~/ $ runspade.py <fully qualified domain name>

If the above command results in the output below (or something very similar) then you have successfully installed and launched the SPADE framework:

SPADE 2.2.1 <gusarba@gmail.com> - http://spade2.googlecode.com

Starting SPADE..... [done]

[info] WebUserInterface serving at port 8008

At this point a browser can be pointed at the localhost (or the "fully qualified domain name" from another network accessible host) on port 8008 similar to that shown in Figure 38.

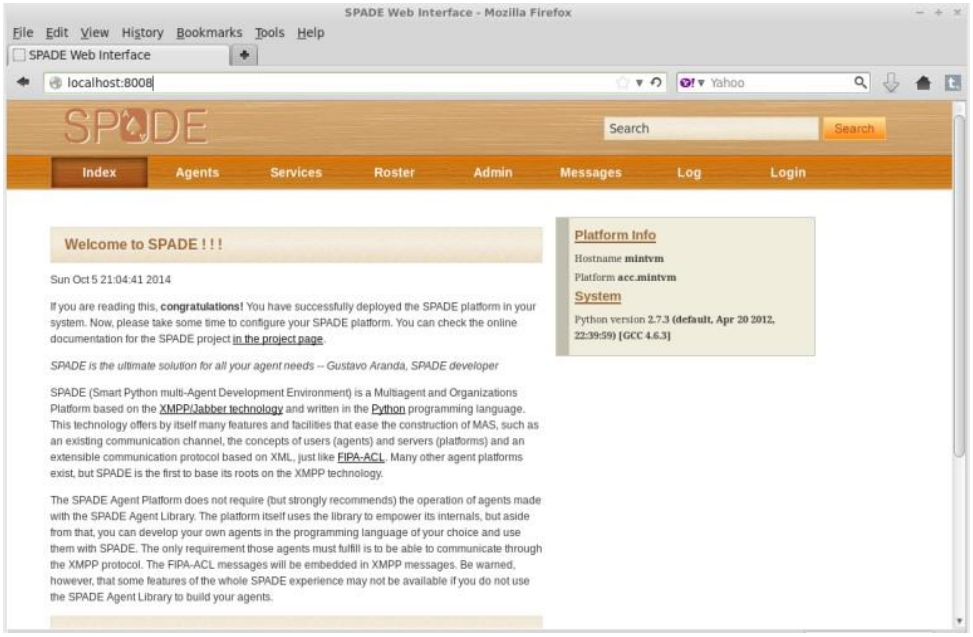

Figure 38. SPADE Framework - fresh install

IX. CURRICULUM VITAE

CHARLES A. FOWLER

[redacted]

BACKGROUND SUMMARY

Over 30 years of progressive experience encompassing:

• Designing/Implementing Enterprise Networks	• Testing Systems End to End
• Conceptualizing Groundbreaking Technologies	• Providing Information Assurance
• Piloting Helicopters (UH-1, AH-1, AH-64)	• Leading Personnel & Testing Efforts
• Product Research, Design & Integration	• Foreign Language Instruction

RESEARCH INTERESTS	*TEACHING INTERESTS*
* Intrusion detection	* Computer Networks
* Data Mining	* Network Security/Information Assurance
* Hybrid Intelligence	* Cyber Operations
* Multi-Agent Systems	* Operating Systems
* Cyber Security	* Databases

CIVILIAN WORK EXPERIENCE

Customer Liaison/Systems Engineer ACES, Columbia, MD FEB 2015 - PRESENT
* Provide systems support for software engineers in several lab environments
* Deploy and maintain product into operational environment
* Liaise between support teams across several organizations – embedded in remote organization 60% hours

Systems Engineer Chesapeake IT, Chesapeake, MD MAY 2014 – JAN 2015
* Provide systems support for a team of software developers in a mixed environment (Linux and Microsoft)
* Maintain two development labs, building & deploying various servers through VMWare
* Install and configure RedHat Directory servers, write customized scripts for numerous administrative tasks

Customer Liaison/Systems Engineer ACES, Columbia, MD JUN 2009 – MAY 2014
* Manage and provide information technology systems support for a team of ~40 software developers in a mixed environment (Linux and Microsoft)
* Maintain development lab with multiple racks of equipment: servers, routers, switches, storage, backup
* Act as liaison between development/leadership teams and customers: gather requirements from diverse customer base providing back and forth feedback as well as clarification of needs and deliverables
* Install and configure systems and software for worldwide customer base
* Conduct hands on training on installation and configuration of custom software to a variety of customers
* Contribute to the writing of user manuals for newly developed software
* Provide daily support for globally fielded products via chat, email, phone and face to face meetings

President/CEO/CIO
ox0xo inc., Annapolis, MD FEB 2001-PRESENT
* Architect, build and maintain information networks for variety of Annapolis area based businesses
* Partner with SAIC in writing proposals for, then participate in award of two Department of Defense Contracts
* Design autonomous underwater vehicle for DARPA's Grand Challenge 2005 & DARPA'S UURC SBIR

Subcontracted to Vandelay Technologies (Ft. Meade, MD) MAR 2012 – PRESENT
* Maintain systems comprising, and provide support for GCCS-I3 (Global Command and Control – Integrated Imagery and Intelligence) application suite, which operates on several 24/7/365 security watch floors.
* Work with new GCCS-I3 feed providers to help gather and define requirements, determine best technological and policy approaches for their integration. Provide documentation & coordinate training to system integrators.
* Perform routine GCCS-I3 system health and status checks, providing daily synopses as well as weekly activity reports to both local and geographically distributed inter-agency affiliates.
* Communicate with globally dispersed feed providers as necessary (weekly at least) to troubleshoot and restore communications and data flows.
* Support myriad GCCS-I3 follow-on customers in diverse environments, including several 24/7 watch floors
* Research and test various technologies and provide reports for optimal way forward (thin-client), to increase robustness of service and accommodate geographically (and numerically) expanding customer base

Subcontracted to CACI (Hanover MD, Taipei Taiwan) APR 2004 – APR 2005
* Architect & build lab network, multiple LAN/WAN comprised of a variety of operating systems, hardware and software including: Trusted Solaris, SunRay, MS Windows Server
* Install and configure Cisco switches, routers and firewalls, Sarvegas, Tivoli, BMC, Managed Objects, Sun Enterprise system
* Support and maintain corporate IT infrastructure and all aspects of phone system in large branch location
* Manage, troubleshoot, and repair production environment IDNX (Integrated Digital Network eXchange - Promina) wide area (global) data-streaming feeds

Systems Engineer/Network Administrator FTI, Edgewater, MD NOV 2001 – JUN 2009
* Lead team efforts to construct testing labs/scenarios, procedures and documentation methodologies suitable for wide variety of products including COTS and GOTS software, and new and modified hardware
* Interim Program Manager for GCCS-I3. Manage team and efforts for performing major systems upgrade, coordinate all players from facilities to a variety of hardware and software vendors.
* Attend quarterly international GCCS-I3 meetings, representing clients interests
* Systems Administrator/Architect for PFN (Processing Forward Network). Provide firewall, router/switch, Citrix, Microsoft and Linux expertise for infrastructure design, build and configuration, information assurance implementation and system testing.
* Technical Team Lead supporting large Focused Demonstration of Capability System FDOC II (Solaris, Linux, Windows, Cyberguard, Oracle, Microstrategy, Foundry, eSecurity, BEA Weblogic etc.)
* Provide vulnerability assessments, forensics, and penetration testing, using a variety of Windows, Linux & Unix based footprinting, enumeration & penetration tools.
* Architect, construct and maintain Citrix and Windows Domain, Active Directory based infrastructure
* Construct lab for testing FOSS (Free & Open Source Software) for malicious behavior, write documentation, and train users

Systems Engineer/Network Administrator
Olympus Group, Alexandria, VA JUL 1998-OCT 2001
* Develop and administer corporate network, NT Domains, Exchange Server, VPN (PPTP), firewalls
Subcontracted to Strategy.com MAR 2000- OCT 2001
* Monitor all aspects of 24/7/365 operations of Strategy.com to include: databases, email systems, proprietary information processing software, remote data feeds & enterprise security
Subcontracted to FMS/U.S. Treasury Dept. OCT 1998-MAR 2000
* Administer load balanced Citrix farm in excess of 20 servers, serving clients worldwide
* Standardize methods for rebuilding, and reconfiguring Citrix and NT Servers

Project Manager
The Network Address, Annapolis, MD APR 1997-JUL 1998
* Integrate internetworking solutions into various environments (wireless LANs, remoting, mainframes, etc.)
* Configure Cisco routers/firewalls, SNMP manageable hubs, CSU/DSUs and other LAN/WAN devices
* Perform wireless site surveys for hospitals, and government clients, specify equipment, layout and security design of 802.11 networks spanning multiple floors, integrated with Novell and Windows architectures.

Sales Representative/ Sales Coordinator JUL 1994-APR 1997
Hertz Equipment Rental Corporation, Forestville, MD
* Initiate & follow up on sales/rental leads, coordinate all aspects of a construction equipment rental business
* Customize and configure CRM software, train and troubleshoot for branch personnel
* Operate and troubleshoot terminal/mainframe connection from branch to Hertz HQ

Arabic Language Instructor Boise Community Education Program, Boise, ID OCT 1993-JUN 1994
* Teach Arabic language, to include: reading, writing, and fundamentals of grammar & Middle Eastern culture

Hard Disk Production Hewlett Packard, Boise Site DEC 1993-MAY 1994
* Design, develop, & implement computerized "human error" tracking system (pareto)
* Represent production component for ISO 9000 certification
* Work with HP proprietary hardware and software systems for product stress testing and test result collation

MILITARY WORK EXPERIENCE
Platoon Ldr and Company Commander (acting) 1-183 Aviation (Attack Helicopter) MAY 1989- JUL 1994
* Command 40 highly trained personnel to accomplish organizational goals and objectives
* Perform Pilot and Co-Pilot Gunner duties in the world's most advanced attack helicopter (AH-64 Apache)

Assistant Operations Officer 1-183 Aviation
* Generate sundry documents for daily operations to include: briefing slides Standard Operating Procedures, various fax headers, calendars & gunnery firing tables.

Battalion Intelligence Officer 1-224 Aviation (Attack Helicopter)
Assistant Cargo Officer 1185th TTU US Army Reserve, Lancaster, PA AUG 1987-SEP 1989
* Port Authority Relations Manager, Ports of Jacksonville, FL & Mobile, AL,

Electronic Warfare, SIGINT Voice Intercept Specialist 311th M I Bn APR 1986-AUG 1987
* Maintain high proficiency in French, Arabic & various dialects for interpretation of tactical language
* Operate and maintain cryptographic equipment and radios (TOP SECRET Security Clearance)

EDUCATION
Doctor of Science - Information Technology, Towson University May 2015
Dissertation Title: A Hybrid Intelligence/Multi-Agent System for Mining Information Assurance Data
Advisor – Dr. Robert Hammell

Masters of Science: Applied Information Technology, Information Security & Assurance
Towson University DEC 2006

B. A. International Studies, Emphasis in Latin American Area Studies MAY 1990
Millersville University of Pennsylvania

US Army Intelligence School SIGINT Voice Intercept (Arabic) DEC 1986
Good Fellow Air Force Base, San Angelo, TX

Defense Language Institute Presidio of Monterey, CA AUG 1986
Arabic& Syrian dialect; qualified in French (DLPT III)

US Army Aviation Officer Basic Course/IERW Ft. Rucker, AL JAN 1992

AH-64*(Apache)* **Aircraft Qualification Course** Ft. Rucker, AL JUN 1992

CERTIFICATIONS, ACTIVITIES, AND AWARDS

Top Secret Security Clearance w/ full scope polygraph	Microsoft Certified Systems Engineer MCSE
Eagle Scout (United States & Canada)	Citrix Certified Professional CCP
Top Gun 1-183 AHB 1993-94 (AH-64 Apache)	Cisco Certified Network Associate CCNA
Order of the Arrow	Cisco Certified Design Associate CCDA

References Available Upon Request

PUBLICATIONS

C. Fowler and R.J. Hammell II, "Mining Information Assurance Data with a Hybrid Intelligence/Multi-agent System", *Proceedings of the 14h IEEE/ACIS International Conference on Computer and Information Science* (ICIS 2015), June 28-July 1, 2015, Las Vegas, NV, accepted.

C. Fowler and R.J. Hammell II, "Converting PCAPs into Weka Mineable Data", *Proceedings of the 15th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing* (SNPD 2014), pp. 47-52, June 30-July 2, 2014, Las Vegas, NV.

C. Fowler and R.J. Hammell II, "Building Baseline Preprocessed Common Data Sets for Multiple Follow-on Data Mining Algorithms " *Proceedings of the 5th Annual Conference on Information Systems Applied Research (CONISAR 2012),* 1-4 November 2012, New Orleans, LA, h*ttp://proc.conisar.org/2012/pdf/2239.pdf.*
Distinguished PhD Student Paper Award

C. Fowler and R.J. Hammell II, "A Hybrid Intelligence/Multi-Agent System Approach for Mining Information Assurance Data,", *Proceedings of the 9th ACIS International Conference on Software Engineering, Research, Management, & Applications (SERA 2011),* pp. 169-170, 10-12 August 2011, Baltimore, MD.