ABSTRACT

Title of Document: ANALYSIS OF A SEMI-SUPERVISED LEARNING APPROACH TO INTRUSION DETECTION.

Benjamin H. Klimkowski, Master of Science
2014

Directed By: Associate Professor Michel Cukier,
Department of Mechanical Engineering

This thesis addresses the use of a semi-supervised learning (SSL) method in an intrusion detection setting. Specifically, this thesis illustrates the potential benefits and difficulties of using a cluster-then-label (CTL) SSL approach to classify stealth scanning in network flow metadata. A series of controlled tests were performed to show that, in certain situations, a CTL SSL approach could perform comparable to a supervised learner with a fraction of the development effort. This study also balances these findings with pragmatic issues like labeling, noise and feature encoding. While CTL demonstrated accuracy, research is still needed before practical implementations are a reality. The contributions of this work are 1) one of the first studies in the application of SSL in intrusion detection, illustrating the challenges of applying a CTL approach to domain with imbalanced class distributions; 2) the creation of a new intrusion detection dataset; 3) validation of previously established techniques.

ANALYSIS OF A SEMI-SUPERVISED LEARNING APPROACH TO INTRUSION DETECTION

By

Benjamin H. Klimkowski

Thesis submitted to the Faculty of the Graduate School of the
University of Maryland, College Park, in partial fulfillment
of the requirements for the degree of
Master of Science
2014

Dedication

To my baby daughter, Penny, whose smile and laughter are priceless.

Acknowledgements

Thank to my advisor, Prof Cukier, for your guidance. One of the hardest things for a mentor to do is to know when not to give directed guidance to a subordinate. Yet Prof Cukier excels in this role. Prof Cukier always allowed me to make my own epiphanies. Even though it felt at times he was giving me enough rope to hang myself, he was always there to prevent me from drowning.

Thank you to my committee members for you time and patience. To Prof Lise Getoor and Prof Dana Nau, I started graduate school primarily interested in systems and security, but after your classes I have gravitated toward more artificial intelligence fields.

Thank you to the working group from the UMD Cybersecurity club, who helped give me different perspectives on how hackers may behave in different contexts. Our dialogue provided valuable insight into this work.

Thank you to all the Soldiers and officers with whom I have served. The list is too long to write.

Thank you to my family. None of this is possible without you. My father, hardest working person I know, my mother the most patient woman I know, and my wife the strongest woman I have ever met.

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

Overview

Computer and network security is at the forefront in the minds of corporate and government leaders. Information technology permeates seemingly every aspect of our lives. Consequently, both public and private sector leaders have invested tremendous amounts of money into developing better protection. Numerous studies have created classification techniques that have asserted promising results. Theoretically, these techniques could be used in an intrusion detection system to identify malicious traffic. Yet, despite considerable investment into research for protection of our computer and network assets, there has been a lack of significant artificial intelligence and machine learning application into real-world intrusion detection. Due to the lack of pragmatic machine learning approaches in intrusion detection, security administrators are limited to signature-based methods and manual processes, which are both error prone and easy to evade. While attackers are growing in sophistication, the security community is struggling to keep pace.

This thesis explores semi-supervised machine learning (SSL) and data mining techniques in the context of network intrusion detection in an effort to address these domain specific challenges. Through a series of controlled and focused experiments, this research attempts to understand how to improve network intrusion detection. Specifically, this thesis focuses on stealth network reconnaissance, a particular subset of malicious activity. By focusing on this specific attack paradigm, we create a cleaner dataset and avoid some of the pitfalls of previous research. Additionally, this research analyzes lightweight network flow data, a form of metadata. Based on the results of this work, we demonstrate that, although there are still some significant challenges, cluster-then-label (CTL) semi-supervised machine learning can be employed with performance comparable to supervised learning in certain settings. That being stated, practical considerations may limit how effective a CTL approach can be in a real world setting.

One of the background motivations for this study was Symons's and Beaver's idea of penetration testing your own network to train a tailor-made SSL classifier [1]. A cost-effective method for developing a tailor-made classifier addresses two specific problems in intrusion detection: the enormous variability among different networks and the rapidly changing nature of attack paradigms. In a practical security setting, a semi-supervised learner would train on a small set of labeled network logs, which a network administrator would audit, as well as the complete set of unlabeled network logs. This training would produce a network specific classifier, which would be sensitive to the particular characteristics and protocols of that network. Furthermore, if this method is proven feasible, then new attack paradigms, for instance a new paradigm that is reported on a hacker convention, could be injected into regular traffic, thus building a classifier that can keep pace with the latest attack trends.

The Threat

Network scanning is an integral part of the attack paradigm; it enables attackers to perform the reconnaissance necessary to identify potential targets and courses of actions to achieve their ultimate objectives. Scanning is typically the first part of the targeting cycle; thus, if a security administrator has warning of scanning, then that administrator should be more focused on the targeted assets (see Figure 1). Furthermore, if there is an indication of scanning that demonstrates a high level of skill on the part of an attacker, then the security administrator should be even more alarmed. Sophisticated and well-funded attackers have the resources to conduct stealthy reconnaissance over the period of days, and often send network probes at such a low rate that it is infeasible for an intrusion detection system (IDS) to detect probing activity within a reasonable time window.

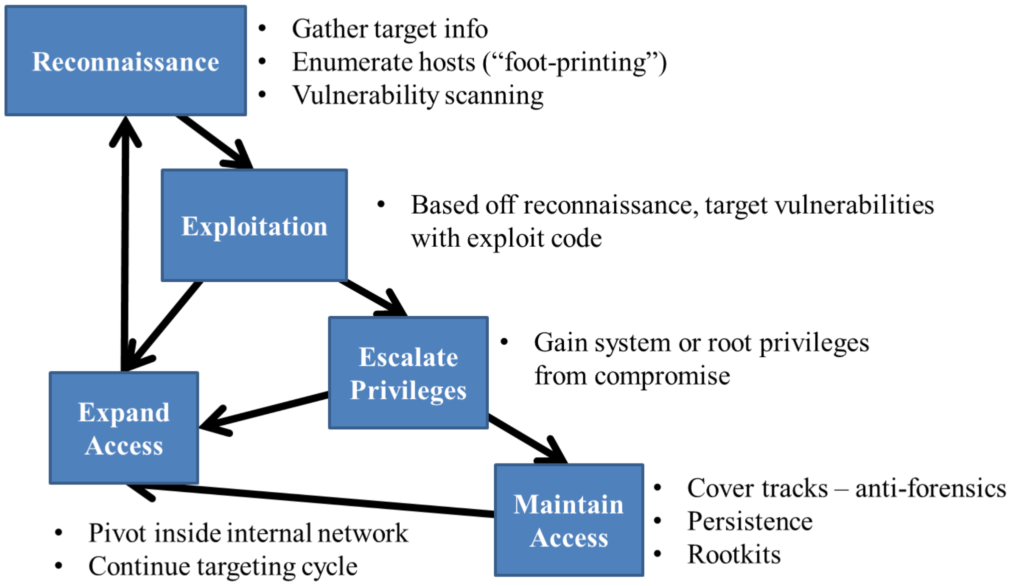

Reconnaissance
• Gather target info
• Enumerate hosts ("foot-printing")
• Vulnerability scanning
Exploitation
• Based off reconnaissance, target vulnerabilities with exploit code
Escalate Privileges
• Gain system or root privileges from compromise
Expand Access
Maintain Access
• Cover tracks – anti-forensics
• Persistence
• Rootkits
• Pivot inside internal network
• Continue targeting cycle

Figure 1: Targeting Cycle

Motivation for Researching Stealth Scanning

While scanning does not directly harm the availability or integrity of a computer system, there are several properties that make it significant. First, detecting scans that last over a long time period is laborious. It is typically a manual process assisted by some signature-based querying. As a result, the process is error prone. Security administrators often ignore scanning due to these challenges. Automated methods that perform better than signatures or other ad hoc techniques can help alleviate this burden. Second, scanning is usually the first part of the targeting cycle. Therefore, if a network defender had knowledge of scanning, the defender would be better able to prioritize and sort subsequent alarms that correspond to activities related to other parts of the targeting cycle. Third, since stealth scanning does not trigger conventional automated methods, such as an IDS, stealth scanning is largely undocumented in existing labeled datasets. In addition, there is a general lack of documentation on how to exactly perform stealth scanning. Finally, scanning is a good starting point to conduct realistic experiments on a live network. Since scanning does not directly harm network resources, experiments can be implemented without extensive investment. Once a technique has been validated for scanning, it can then be applied to more intrusive portions of the targeting cycle.

Motivation for Studying Network Flow Metadata

Network flow metadata is important for several reasons. First, network flow data is ubiquitous. Almost all routers produce network flow records. In certain situations, a security administrator performing network forensics may only have network flow data available to analyze. Second, network flow data requires order of magnitudes less storage than other network log data such as packet capture (PCAP). This smaller footprint means logs can be stored for a longer period. Finally, the more network protocols use encryption, the more network defenders will need techniques that can infer attacks without relying on deep packet inspection of packet’s payload.

Objectives

This thesis aims to address the following questions:

1. Does previously established scan detection techniques perform well on a specific set of malicious data (stealth scanning) produced on a real production network? This question aims to determine the strengths and weaknesses of previous approaches, and to gauge how the network environment affects the techniques. If a previous approach works well, it can be used as a basis of comparison for new techniques.
2. Do semi-supervised methods perform comparably to supervised methods? The objective of this question is to show whether semi-supervised methods can perform as well or potentially better than supervised methods, while using a fraction of the required effort.
3. Do any of these techniques show promise in a practical network intrusion detection setting? As this thesis will show, network intrusion detection has some unique challenges that require specific attention in order for an implementation to be successful.

Experimental Overview

Figure 2 illustrates the main phases of the study. Before any analysis of a classifier could be performed, this study had to develop and catalog attack paradigms, build a dataset, and audit that dataset for errors. After the data was properly audited and organized, the scan detection methods were evaluated.

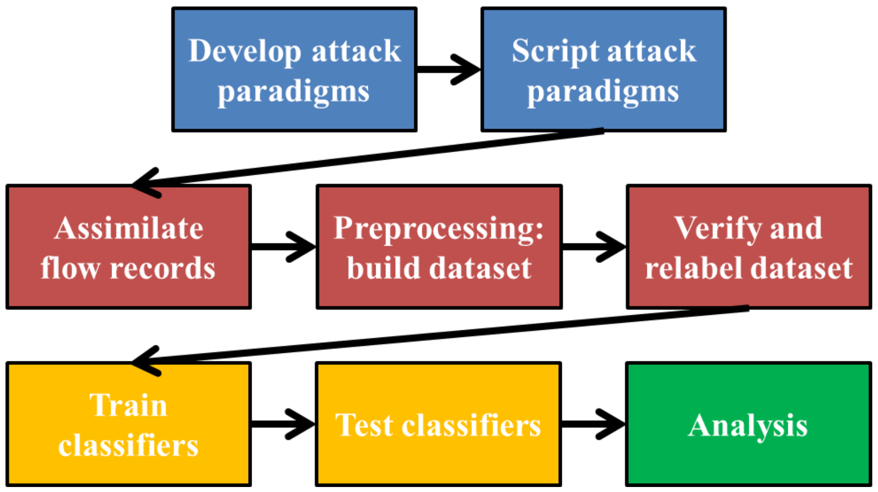


Figure 2: Experimental Phases

Contributions

There are three main contributions to this research. First, it validates previous work. Too often intrusion detection studies are neither validated nor replicated and this lack of scientific rigor results in limited real world applicability. This lack of validation is further exacerbated by the changing nature of network technology. What may work today, may be ineffective or impractical tomorrow. Second, this work has produced a clean, labeled dataset that was built in a transparent and controlled manner. There is a critical lack of labeled datasets in intrusion detection. Furthermore, this thesis clearly lays out how the experiment was conducted so that other researchers can perform similar studies in an effort to replicate and reproduce results. In addition, steps have been taken to make this dataset publically available. Finally, to our knowledge, this thesis is the first dedicated study on using CTL SSL techniques to detect a specific subset of malicious activity, and one of the few studies on SSL in intrusion detection. The results of this study document potential issues with a SSL implementation in intrusion detection and other application domains where there is an imbalance in class distributions.

Organization

The remainder of this thesis is organized as follows. In Chapter 2, relevant background concepts and previous works are explained. Chapter 3 describes the methods employed in this experiment. Chapter 4 provides an overview of the dataset. Chapter 5 describes the experimental parameters for the trials. Chapter 6 analyzes the trial results with insight to practical application issues. Chapter 7 summarizes this thesis and provides recommendations for future work.

Chapter 2: Background and Related Works

Network FlowData

Network flow data is metadata about network based transactions between pairs of endpoints in a network. The heart of network flow data is the concept of an IP flow, which is a set of similar packets observed on a certain point and time in a network, going from one source host to one destination host. A flow record summarizes the pertinent characteristics from the IP flow (see Table 1). Typically, a router will record IP flow records and then export them to a server for storage. For most network transactions between two endpoints, the transaction is bidirectional, so the router or observation point will record two flows. For instance, when a client browses to a web server, a router collecting network flows will record two flow records: one that describes the set of packets from the client to the server and one in the reverse direction. A network flow log is a flat collection of network flow records. Administrators use network flow data for a variety of purposes including billing, network monitoring, capacity planning, security analysis, and data mining [2].

Table 1: Basic Network Flow Record Information

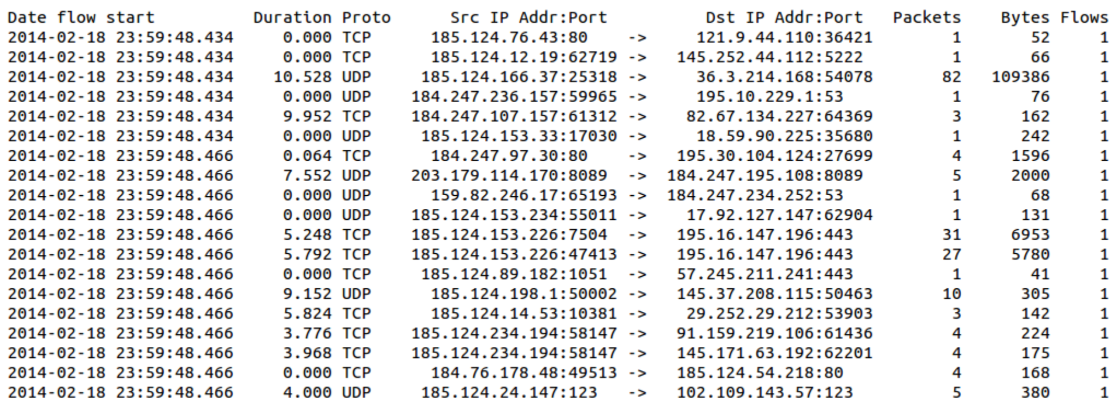
Field	Remarks
Time	Timestamp of when flow began
Duration	The length of the flow in milliseconds
Source IP address	The TCP/UDP port number of the source socket
Source port	
Destination IP address	
Destination port	The TCP/UDP port number of the destination socket
OSI Layer 3/4 Protocol	Layer 3 Protocol used. For IPv4, typically the layer 4 protocol is specified (TCP/UDP)
Router interface	The interface on which the packet entered
TCP Flags	If TCP, this field is the union of all the TCP flags seen
Number of packets	
Number of bytes	

Some aspects of network flow data are important to understand in order to analyze network communications. A flow record is created whenever the observing device sees a unique IP address port combination. Thus, a single communication graph between two endpoints could have multiple flow records if the network communication involves multiple layer 7 protocols (i.e. spans multiple ports). The observing device records what it sees from the packet's header and typically does not have any enhanced security features to detect spoofing. Therefore, security analysts need to be cognizant that attackers may have the ability to obfuscate themselves in certain attacks. In addition, a router may produce a flow record if any of the conditions in Table 2 are met. As a result, proper analysis must account for premature segmentation of flows.

Table 2: Conditions for Network Flow Expiration

Condition	Remarks
TCP flag received	If a host sends a FIN or RST flag on a TCP connection, the flow will be terminated [2].
Inactivity	If the flow is inactive for some predefined threshold, the flow is terminated [2]. Default for NetFlow is 15 seconds [3].
Long flows	If the flow continues to be active longer than a prefunded threshold, the flow is terminated [2]. Default for NetFlow is 30 minutes [3].
Memory exhaustion	Depending on the implementation, a router may prematurely close a flow record, if it is low on memory. By default NetFlow will expire 30 flow records prior to its cache hitting maximum capacity [3].

It is worth noting that the term “network flow” refers to a family of similar protocols. Cisco pioneered the field with their priority format NetFlow. NetFlow version 9 is the basis for the open IETF standard IP Flow Information Export (IPFIX) protocol. Despite the existence of a multitude of priority formats, each format is similar enough that a technique developed on one will tend to work with other vendor formats. In other words, the network flow techniques will work on all formats as long as the technique is limited to the information in Table 1 and it does not rely on any user-defined fields. This thesis used NetFlow version 5 through the nfdump toolset. An example output from nfdump is show in Figure 3.



```
Date flow start          Duration Proto      Src IP Addr:Port          Dst IP Addr:Port   Packets    Bytes Flows
2014-02-18 23:59:48.434     0.000 TCP     185.124.76.43:80    ->     121.9.44.110:36421          1       52     1
2014-02-18 23:59:48.434     0.000 TCP     185.124.12.19:62719 ->   145.252.44.112:5222           1       66     1
2014-02-18 23:59:48.434    10.528 UDP    185.124.166.37:25318 ->     36.3.214.168:54078         82   109386     1
2014-02-18 23:59:48.434     0.000 UDP   184.247.236.157:59965 ->     195.10.229.1:53             1       76     1
2014-02-18 23:59:48.434     9.952 TCP   184.247.107.157:61312 ->    82.67.134.227:64369          3      162     1
2014-02-18 23:59:48.434     0.000 UDP    185.124.153.33:17030 ->     18.59.90.225:35680          1      242     1
2014-02-18 23:59:48.466     0.064 TCP     184.247.97.30:80    ->   195.30.104.124:27699          4     1596     1
2014-02-18 23:59:48.466     7.552 UDP   203.179.114.170:8089  -> 184.247.195.108:8089           5     2000     1
2014-02-18 23:59:48.466     0.000 UDP     159.82.246.17:65193 -> 184.247.234.252:53             1       68     1
2014-02-18 23:59:48.466     0.000 UDP   185.124.153.234:55011 ->    17.92.127.147:62904          1      131     1
2014-02-18 23:59:48.466     5.248 TCP   185.124.153.226:7504  ->   195.16.147.196:443           31     6953     1
2014-02-18 23:59:48.466     5.792 TCP   185.124.153.226:47413 ->   195.16.147.196:443           27     5780     1
2014-02-18 23:59:48.466     0.000 TCP    185.124.89.182:1051  ->   57.245.211.241:443            1       41     1
2014-02-18 23:59:48.466     9.152 UDP     185.124.198.1:50002 ->   145.37.208.115:50463         10      305     1
2014-02-18 23:59:48.466     5.824 TCP     185.124.14.53:10381 ->    29.252.29.212:53903          3      142     1
2014-02-18 23:59:48.466     3.776 TCP   185.124.234.194:58147 ->   91.159.219.106:61436          4      224     1
2014-02-18 23:59:48.466     3.968 TCP   185.124.234.194:58147 ->   145.171.63.192:62201          4      175     1
2014-02-18 23:59:48.466     0.000 TCP     184.76.178.48:49513 ->   185.124.54.218:80             4      168     1
2014-02-18 23:59:48.466     4.000 UDP    185.124.24.147:123   ->   102.109.143.57:123            5      380     1
```

Figure 3: Example Nfdump Output

Scanning

This research focuses on port scanning, a specific subset of network reconnaissance. Scanning covers a range of activities by which attackers attempt to gain information about a target network and its hosts. Attackers typically perform initial discovery of hosts through ping scans (“ping sweeps”), reverse-DNS resolution, and ports scans. Port scans attempt to reach open TCP and UDP services on hosts. In the case of TCP scans, various header flags may be set to gauge how the network and hosts are configured to respond. After initial scans, an attacker may employ more interactive scripts to ascertain specific vulnerabilities in protocols or how the server is implemented. This activity, known as vulnerability scanning, varies in level of how invasive it is to the host machine and has a different traffic profile from port scanning. In addition to information about services hosted, attackers can use ping scans, port scans, and vulnerability scans to infer OS version, network configuration and other information [4].

While scanning can take many forms, most researchers categorize port scans into two basic categories based on the scan’s intended target footprint: horizontal scans and vertical scans [5]. Horizontal scans refer to scans where an attacker seeks to gain information on a range of hosts on a network: which ones are accessible, what services they hosts, what version of a protocol, etc. Vertical scans refer to scans where an attacker seeks to gain information about a specific host. These scans typically include a greater range of ports and protocols targeted, and they may include other reconnaissance techniques to infer OS build, poor protocol configurations, etc.

An understanding of each scan type and their purpose provides insight on how available network record sources can be processed and minded to detect this activity.

Challenges with Machine Learning in Intrusion Detection

Machine learning, despite successful application in other areas, has had difficulty in being adopted in practical network intrusion detection (NID) settings. Many studies fail to recognize fundamentally how machine learning algorithms work and their underlying assumptions. Furthermore, some unique aspects of classifying network traffic make it a dramatically different task than other applications such as classifying spam or optical character recognition (OCR). In addition, intrusion detection has the added problem of an adversary community that constantly seeks ways to tune attacks and evade detection [6]. Understanding these idiosyncrasies is critical to understanding how to apply machine learning techniques more effectively.

One of the most frequent flaws in machine learning applied to NID is assuming that novel attacks will be detected given a large dataset of known activity [6]. On the surface, machine learning algorithms can classify instances as “abnormal” vs. “normal” or “malicious” vs. “benign”; however, the algorithms require “abnormal” or “malicious” training experience to develop a classifier that generalizes well. If the datasets on which the algorithm is trained does not contain representative samples of attacks, then the resultant classifier may be inadequate. As Sommer and Paxson assert, studies often wind up “training an anomaly detection system with the opposite of what it is supposed to find…it requires having a perfect model of normality for any reliable decision” [6]. This idea is referred to as the “closed world assumption”; the idea of specifying only positive examples and adopting a standing assumption that the rest are negative [6]. As a consequence of this misunderstanding, the trained classifiers often do not generalize adequately in future settings.

There are some particular aspects of NID that can make a naive implementation of a machine-learned classifier ill-posed. NID is an area of study that is plagued by the base-rate fallacy [7]. Since the proportion of benign traffic is order of magnitudes larger than the proportion of malicious traffic, many studies fail to highlight that while the accuracy of a detection scheme is ostensibly high, for instance 95%, its performance will be poor in practice. This apparent incongruent is due to the detection scheme generating orders of magnitude more false positive alarms than true positive alarms. Consequently, in order for a detection scheme to be of practical use, it must detect at an extremely high accuracy, which for a learning algorithm may require an infeasible amount of training data and may not generalize outside of the network on which the learning algorithm trained.

The costs of classification errors in NID are much higher than in other domains. Other areas where applications have been successful demonstrate error tolerance (like recommending products for e-commerce). In these domains, making a classification error has a negligible impact. In NID, the impacts of errors are extreme; false positives can result in a significant waste of time, false negatives in compromised computer systems [6]. Alerts for scanning in particular are full of false positive alarms. The base-rate fallacy further exacerbates the situation, because users of a system will quickly lose faith if they waste all of their time chasing large numbers of false alarms without ever finding a true positive.

In other machine learning applications, like spam, it is easy to correct/validate false positives or false negatives. Other successful applications have been able to improve their performance over time because the user base can easily provide feedback. Subsequently, this feedback can tune classifiers and correct datasets. Intrusion detection alerts are inherently difficult to evaluate [6]. Even when classification errors are discovered, it is not in a volume that is large enough to significantly improve a classifier performance or build a cleaner dataset.

Spam classification also demonstrates that certain machine learning applications can benefit from an unbalanced cost mode. In some cases, designers can gear an algorithm to err towards false negatives to avoid the more adverse cost of a false positive. This skewing allows a machine-learned classifier to be employed in a real-world setting without burdening the user. NID does not offer the opportunity for such a tradeoff. Both false positives and false negatives are extremely undesirable [6].

Network traffic demonstrates an enormous variability of benign traffic. As Sommer and Paxson assert, it is "difficult to find stable notions of normality." Many of the traffic profiles are heavy-tailed distributions, where there are events that are significantly far away from the mean but only occur at infrequent times. This means studies seeking to develop an accurate probability distribution would require an enormous amount of time and effort. The rapidly changing nature of network configurations would further impede a meaningful study. What the network looks like now, may not be what the network looks like after the installation of a new service, device, or a major connectivity change. Furthermore, what is normal at one site, may not be normal at another site. A study that does not consider how the experimental network impacts the classifier will be doomed to low adoption. Without a meaningful discussion on the nature of the network traffic, a scheme could fail to demonstrate the same classification accuracy or may not even be able to be implemented [6].

Many machine learning studies fail to account for how attackers operate. Sophisticated attackers study security research and develop ways to tune their tactics to evade detection. This antagonistic relationship makes naive studies irrelevant when attackers can arbitrarily adjust their traffic characteristics that will be present in log data. For instance, an attacker may pad their packet payloads so that the packet size appears less suspicious. The padding does not interfere with the attack, but if a classifier has been trained to look for instances of small packet sizes, the modified packet may evade detection. Therefore, any study needs to develop a classification scheme from the motivation and perspective of an attacker. Ideally, the classification scheme needs to choose features that are invariant for a type of attack, so the attacker cannot easily adjust his/her technique. Consideration should also be given on how a scheme could be defeated. Even if an attacker can evade one security measure, it may present an opportunity for alternate security measures or methods to be used to detect the attack.

Challenges with Intrusion Detection Datasets

Arguably, the most significant challenge to intrusion detection research is the lack of sound publically available labeled datasets. One reason for the absence is the difficulty in classifying and verifying network traffic to build the dataset in a clean manner. As Sommon and Paxson assert the investment of building an experimental set-up is often more difficult than developing the detection scheme itself [6]. For studies that built their own sets, privacy and security concerns typically prevent researchers from sharing their information with the community [8]. Currently, there are three publically available labeled datasets: the 1998/1999 DARPA Lincoln Lab Intrusion Detection Evaluation dataset (DARPA-98/99), the Knowledge Discovery and Data Mining Cup 1999 dataset (KDD-99), and the Kyoto University 2006-2009 Honeypot dataset (Kyoto2006+). Many critical studies have shown these datasets have considerable issues, to include out-of-date attack paradigms, unrealistic traffic modelling, ambiguous labeling schemes, erroneously labeled data, and a significant lack of validation [1, 9, 10, 11, 12, 13].

The DARPA-98/99 and the KDD-99 are the two most commonly used datasets, and they both have significant issues. Both datasets are over ten years old; network bandwidth, applications and attacks have all changed significantly since the original studies. Both the DARPA-98/99 and KDD-99 used simulated traffic that is supposed to be representative of a typical Air Force base. Critical studies of these datasets have demonstrated, however, that the traffic is not representative of even the installation that it is supposed to model [9, 10]. Despite the obvious deleterious effects unrealistic simulations could have, the original DARPA study did not perform analysis on how the artificial nature of the simulated traffic affected their evaluation of various IDSs. Many studies perpetuate this mistake and report findings without consideration of this glaring issue. As McHugh asserts, “the burden is on the experimenter to show that the artificial environment did not affect the outcome of the experiment.” Furthermore, the categories of malicious data used in the datasets are often ambiguous. The broad, attack-centric taxonomy does fail to describe clearly from the IDS's perspective what is alarming. Tavallaee et al. demonstrated a serious lack of validation in both datasets that has led to errors, such as redundant records, dropped traffic records, etc. While these two datasets are the most widely used and therefore the most criticized, it is worth noting that many of the homegrown datasets in intrusion detection studies repeat many of the same errors with respect to modelling, analysis, and validation [9].

Kyoto2006+ is a relatively new dataset that took an innovative approach in an effort to capture more realistic attack traffic. In contrast with the scripted nature of the traffic in DARPA-98/99 and KDD-99, Kyoto2006+ used honeypots to collect real attack information and it injected traffic from real servers (a mail and DNS server) to create the non-attack traffic. In order to label the traffic, the creators of the data used a combination of a network IDS, host anti-virus and a shellcode detection tool known as Ashula [10]. The use of the Ashula tool offers an interesting aspect in the Kyoto2006+ labeled set: it allowed the researchers to demonstrate how effective their detection scheme is at detecting malicious traffic for which traditional IDS and host anti-viruses did not have a signature. This concept supports the underlying hope behind many machine learning studies that the classifier can generalize to detect new and previously unseen forms of malicious data.

There are significant issues with the Kyoto2006+ dataset. First, the manner in which the creators injected normal traffic is problematic. They assumed that all injected traffic can be classified normal because they "observed that there is few [sic] attack data even if the server has received cyber attacks" [10]. As we will demonstrate later in this paper, this assumption can have disastrous effects depending the researcher' activity of interest and the learning technique which they apply to the dataset. Furthermore, the proportions of injected traffic and type of inject traffic is completely unrealistic. The authors of the dataset report 50,033,015 normal network transactions and 42,617,536 malicious ones. As previously discussed, in reality the actual amount of normal traffic is orders of magnitude more than the malicious traffic. The "normal traffic" is limited to two hosted services and the management traffic to the servers. It lacks any client behavior such as peer-to-peer that could have serious effects on a classification scheme. These issues mean that any parametric learning approach will fail to generalize to any real-world network. In addition to labeling issues, there is no consideration on how sophisticated attackers act on honeypots. There is research among the attack community on how to detect honeypots and ways to exploit them in a stealth manner [12, 11]. This lack of consideration means that the automated architecture may fail to detect certain hackers, and would incorrectly label certain network transactions as benign. It also undermines the previous assumption that the captured traffic is representative of all of the possible attacks that the network will see in the "wild." Another issue is that the Ashula tool is no longer available for download, and its parent website, www.secure-ware.com, is down. Their 2009 paper does not discuss Ashula' inner workings. Without any conversation on how Ashula works, situations in which it may not detect a type of exploit, or how it can be defeated, researchers cannot accurately claim that it detects all instances. Finally, other papers that used the Kyoto2006+ dataset have demonstrated that it may contain errors [1]. Without an enduring effort at curating the dataset, when a researcher finds a labeling error, there is no mechanism to disabuse the dataset for future studies.

Previous Semi-Supervised Learning Work

Semi-supervised learning techniques are an emerging area of research in intrusion detection. In one of the earliest works, intrusion detection is modelled as a partially observable Markov decision-making process (POMDP) and uses a semi-supervised approach (Expectation Maximization to the learn conditional probability distributions) in order to classify legitimate and misuse user behavior in a UNIX terminal [12]. In [13], a co-training method for intrusion detection is applied to the KDD-99 dataset. In [1], the authors explored the use of non-parametric graph-based methods (Laplacian Eigenmaps and Laplacian Regularized Least Squares) on the Kyoto2006+ dataset. One of the promising aspects of their approach is that the authors do not have to make any assumptions about the probability distributions of the traffic. The paper also made an assertion that semi-supervised techniques could be used in practical real world settings, where a system administrator could build his/her own mixed labeled dataset by auditing a portion of existing normal traffic and performing some penetration testing. This comment is one of the motivations for this study.

Cluster-then-label (CTL) SSL techniques have been applied to other domains with success. The authors in [14] used CTL techniques successfully in three-dimensional character recognition, and [15] demonstrated success in classifying eight datasets from the UC-Irvine Machine Learning Repository. [16] provided a theoretical analysis on situations where a CTL approach provides a better error bound than a pure supervised classifier, and situations where CTL techniques cannot perform any better. At the time of this thesis, there appears to be no application of a CTL approach to intrusion detection.

Previous Scan Detection Work

Numerous supervised learning and data mining studies have been applied to scanning. [17] developed the Stealth Probing and Intrusion Correlation Engine (SPICE), which uses a Bayes network approach to compute the probability that a network flow record is a scan. [18] developed a method based on sequential hypothesis testing (SHT). Their scheme, Threshold Random Walk, evaluates a probability ratio for each connection $\frac{\Pr_{\text{scanner}}}{\Pr_{\text{non scanner}}}$ per remote host, and performs SHT until it has seen enough connections to classify the host as a scanner or non-scanner based on two predefined thresholds. [19] proposed innovative clustering and mining technique to visualize scans and attack data.

Simon et al., from UMN MINDS, presented an innovative data-mining approach to scan detection [20]. Their approach performs heavy preprocessing of network flow records, transforming them in to a dataset that consists of rich features before using a supervised learner. These features, such as the number of distinct internal IP addresses touched by a single external IP, intuitively correspond to characteristics of scans that are useful in distinguishing them from non-scan traffic. After preprocessing, the authors train a rule-learning algorithm known as Repeated Incremental Pruning to Produce Error Reduction (RIPPER). This thesis replicates their working using the same feature set and classifier in order to gauge the effectiveness of their method and use it as a benchmark to evaluate the semi-supervised technique.

Chapter 3: Method

Supervised Classifier

Rule-based algorithms, which follow an algorithmic paradigm similar to decision trees, are an easy to use family of algorithms for inductive inference. In essence, these rule-based algorithms create a sequence of rules, where each rule attempts to cover and separate out as many instances of one class of data as possible. Different versions of algorithms vary in the heuristics that they use to select rules and the logic to prune rules to avoid overfitting. The most significant drawback of rule-based algorithms is their tendency to overfit to the dataset on which they were trained [21].

RIPPER is one of the most popular rule-based classifiers, due to optimizations for fast runtime and pruning logic that minimizes overfitting effects [22]. It runs in $O(n(\log n)^2)$ time, opposed to the popular decision tree algorithm C4.5, which runs in $O(n^3)$ time [21]. The resultant classifier RIPPER produces is intuitive to the human reader; it essentially is a series of "if-then" statements, which could be used in existing signature-based IDS. RIPPER can handle datasets that are not linearly separable. It does not require a priori knowledge of the underlying statistical distribution of the dataset. This fact allows research to avoid making erroneous assumptions. In addition, it is robust to noise, both errors in classifications of the training instances and errors in the feature values that describe these instances [21]. This property offers the user some latitude since building a flawless training set in NID is near impossible. The pseudocode in Figure 4 explains RIPPER operations. Figure 5 explains the metrics and heuristics, which RIPPER uses.

```
function BuildRuleSet($P, N$)
$RuleSet$ = { }
$DL \leftarrow$ DescriptionLength($RuleSet, P, N$)
/* Split $P$ and $N$; two sets at a ratio of 2:1                    */
$GrowSet,\ PruneSet \leftarrow$ split(P,N)
while $P \neq 0$ do
    $Rule \leftarrow$ GrowRule($GrowPos, GrowNeg$)// Uses info-gain heuristic
    $Rule \leftarrow$ PruneRule($Rule, PrunePos, PruneNeg$, heuristic=$W$)
    if $DescriptionLength(RuleSet, P, N) > DL + 64$ or $Error\ rate > 50\%$
    then
        Delete $Rule$ from $RuleSet$
        break
    $DL \leftarrow$ DescriptionLength(RuleSet,P,N)
    remove all $P, N$ covered by $Rule$
return $RuleSet$
end function
```

```
function OptimizeRuleSet($RuleSet, P, N$)
for each Rule $R \in RuleSet$ do
    remove $R$ from $RuleSet$
    /* A new permutation of a 2:1 split                                 */
    $GrowSet,\ PruneSet \leftarrow$ split(P,N)
    $\forall P, N \in PruneSet$ if $RuleSet - R$ covers $P, N$ then
        remove $P, N$ from $PruneSet$
    /* Build competing rules from the new split data                    */
    $R1$ /* Build R1 from scratch                                       */
    $R1 \leftarrow$ GrowRule($GrowPos, GrowNeg$)// Uses info-gain heuristic
    $R1 \leftarrow$ PruneRule($R1, PrunePos, PruneNeg$, heuristic=$A$)
    /* Build R2 adding antecedents to R                                 */
    $R2 \leftarrow$ GrowRule($R, GrowPos, GrowNeg$)
    $R2 \leftarrow$ PruneRule($R2, PrunePos, PruneNeg$, heuristic=$A$)
    /* Choose the rule that has the minimum DL                          */
    $R \leftarrow$ FindMinDL(R,R1,R2)
    remove $P, N$ covered by R
if $P \neq \{\}$ then
    BuildRuleSet(P,N)
return $RuleSet$
end function
```

```
function RIPPER($M_l$,$k$)
$M_l \leftarrow$ labeled data set
$k \leftarrow$ rounds of optimization to perform
$P \leftarrow$ positive training instances (minority class)
$N \leftarrow$ negative training instances(majority class)
$RuleSet \leftarrow$ BuildRuleSet(P,N)
repeat
    $RuleSet \leftarrow$ OptimizeRuleSet($RuleSet$,$P$,$N$)
until $k$ times ;
return RuleSet
end function
```

Figure 4: RIPPER Pseudocode (*adapted from* [21] [22], [23])

$$\text{Description Length} = \frac{1}{2}\left[\lceil \log_2 k \rceil + k \log_2 \frac{x}{k} + (x-k)\log_2 \frac{1}{1-\frac{k}{x}}\right]$$

$$\text{Information-gain} = p\left[\log_2 \frac{p}{t} - \log_2 \frac{P}{T}\right]$$

$$A = \frac{p+n'}{T}$$

$$W = \frac{p+1}{t+2}$$

$k =$ number of conditions in the rule

$x =$ number of possible conditions in the rule

$p =$ number of positive examples covered by this rule (true positives)

$n =$ number of negative examples covered by this rule (false negatives)

$t = p + n$; total number of examples covered by this rule

$P =$ number of positive examples in the set before the rule is implemented

$N =$ number of negative examples in the set before the rule is implemented

$T = P + N$; the totatl number of examples in the set before the rule is implemented

$n' = N - n$; the number of negative examples not covered by this rule (true negatives)

Figure 5: RIPPER Metrics and Heuristics (*Adapted from* [22], [21])

Unsupervised Clustering

The unsupervised clustering method used in this thesis was k-means++. K-means++ is an optimized version of k-means. It uses randomized seeding to increase runtime performance (empirical studies have shown a performance speedup by factor 2 to 10). While finding the optimal clustering is NP-hard, k-means++ is guaranteed to be $O(\log(k))$ competitive to the optimal solution [24].

Semi-Supervised Method

The experiments conducted in this thesis depart from the mainstream SSL approaches, which are either graph-based or based on a parametric mixture model, and explores “cluster-then-label” (CTL) SSL methods. The overall intuition and assumption is that in a given domain similar instances tend to group together. In this paradigm, first an unsupervised algorithm groups data points into clusters known as decision sets. Next, a supervised learning algorithm per cluster is trained on the labeled instances. This supervised classifier is then transductively applied to the unlabeled instances within the decision set. There are two ways to classify future instances. First, a CTL algorithm can map an instance to a decision set, then apply that set's supervised classifier. Alternatively, the CTL algorithm can use a global classifier. This global classifier is built after performing an additional iteration of supervised learning during the build portion of the algorithm (see Figure 6). This thesis evaluated both variants, primarily focusing on the former.

```
function BUILD_CTL($M$,k,f$_{sl}$,$c_{cluster}$)
    • $M \leftarrow$ mixed label set
    • $k \leftarrow$ the desired number of clusters
    • $f_{sl} \leftarrow$ supervised learning algorithm
    • $c_{cluster} \leftarrow$ unsupervised clustering algorithm

Let $M_l$ be the labeled portion and $M_u$ be the unlabeled
1. Train the a default global classifier, $f_{SL_{default}} \leftarrow train(M_l)$
2. Perform clustering, $C \leftarrow c_{cluster}(M, k)$
3. Build cluster classifiers
for cluster $i \in C$ do
    if $M_{l,i} \neq \emptyset$ and $f_{SL_i}$ has enough labeled data then
        Train a supervised classifier, $f_{SL_i} \leftarrow train(M_{l,i})$
    else
        if $M_{l,i} \neq \emptyset$ then
            Use majority vote to label the remaining data,
            $f_{SL_i} \leftarrow majority(M_{l,i})$
        else
            $f_{SL_i} \leftarrow f_{SL_{default}}$
    Label the remaining unlabeled data,
        $M_i' \leftarrow f_{SL_i}(M_u, i)$
if retraining then
    Combined the modified labeled dataset from all clusters,
    $M_l' \leftarrow \bigcup_{i=1}^{n=k} M_i'$
    Train supervised learner on the modified labeled dataset,
    $f_{SL_{retrain}} \leftarrow train(M_l)$
end function

function CLASSIFY_CTL($x$, retraining)
$x \leftarrow$ data instance
if retraining then
    return $f_{SL_{retrain}}(x)$
else
    Find cluster membership, $c_i$, for $x$
    return f$_{SL_i}(x)$
end function
```

Figure 6: Cluster-Then-Label Algorithm

CTL offers several advantages over the other approaches. First, although CTL is similar to mixture-models, it is fundamentally more general. It makes no assumption what the underlying distribution is for the populations of benign and malicious. As Zhu and Goldberg assert, if a poor model is assumed it could have deleterious impact on classification [25]. Hu et al. demonstrated that network traffic and in particular malicious traffic can be heavy-tailed, which may exacerbate poor choices in models [26]. In addition, since this experiment injects malicious traffic, any supposition on a distribution would be fundamentally flawed and skewed from the true distribution.

While there may be some similarities in mathematical reasoning between CTL decision sets and the manifold regularization of graph-based methods, the clustering methods employed in the CTL approach are mathematically simpler and computationally less expensive than the graph-based methods. As Goldberg et al. suggest, it follows a “wrapper” design paradigm, where multiple and previously established techniques can be used for the unsupervised and supervised steps of the algorithm without having to develop newer or more confusing techniques [14]. Since many of the machine learning and SSL algorithms are computationally expensive, a wrapper design paradigm allows the choice of optimized supervised and unsupervised implementations. This flexibility enables CTL to process large datasets faster than other algorithms that have a greater runtime complexity.

The need for an easy to interpret classifier goes beyond mere convenience. Some techniques like artificial neural networks (ANN) or support vector machines (SVM) may not offer much insight into what causes a particular instance to be detected. It would be fallacious to subscribe to a “black box” mentality that assumes that once a classifier is in place it will continue to exhibit the same success rate.

Without insight into how something works, attackers could potentially run many variations of an attack against the same “black box” until they discover an undetected permutation. Choosing a classifier that supports anti-forensic analysis is paramount.

Cluster-then-label Analysis

The purpose of this section is to provide theoretical background on situations where the CTL approach has a provably better error bound than a pure supervised approach. In particular, this section will summarize Singh, Nowak and Zhu’s analysis of CTL error convergence rates. This section will also highlight some of the implicit assumptions of the model that are subtle but have tremendous impact if not properly followed.

Error Performance

Singh et al.’s analysis provides the mathematical justification to why empirical evidence demonstrates that CTL appears to improve error performance in some cases but not in others. Their core idea is based on a geometric understanding of how the labeled and unlabeled data are distributed. In essence, if the individual clusters, which make a CTL decision set, are more discernable than the whole set of labeled data, then CTL SSL can yield higher accuracy rates than a pure supervised classifier. Figure 7 illustrates this concept; in portion in Figure 7a presents an omniscient view of two classes that exist in discernable dense clusters. Unfortunately, the amount and distribution of the available labeled data in Figure 7b is not enough to accurately distinguish the boundary between the two classes. Figure 7c shows that when unlabeled data is combined with labeled data, the boundary becomes distinct.

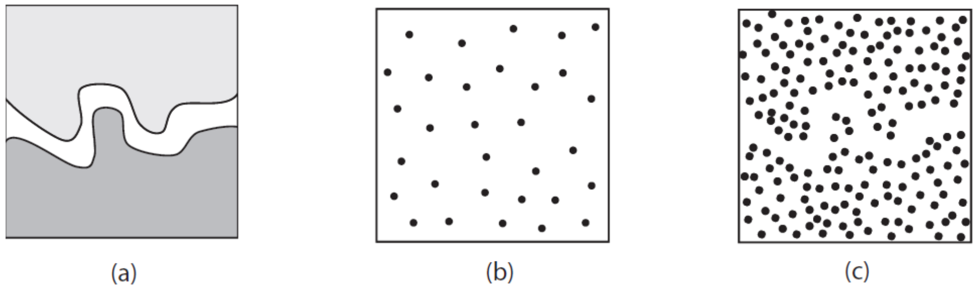


Figure 7: a) Omniscient view of two classes, b) the whole set of labeled instances, c) the set of labeled and unlabeled instances (*Adapted from* [16])

Singh et al.'s proof consists of two major parts. In the first part, Singh et al. establish a lower bound on the separation distance between clusters in order for data points to be sufficiently distinguishable; i.e. margin necessary to cluster instances with high probability ($O(m - \frac{1}{m})$) into the correct decision set without any additional knowledge. Figure 8 shows this bound, which relates the margin between clusters, γ, to the average density within a cluster.

$$|\gamma| > c\sqrt{d}\left(\frac{m}{\log^2 m}\right)^{-1/d}, \text{ where}$$

$$\begin{aligned} \gamma &:= \text{ the margin} \\ c &:= \text{a positive constant}, > 0 \\ m &:= \text{ number of labeled instances} \\ d &:= \text{ the number of dimensions (features)} \end{aligned}$$

Figure 8: Margin-cluster Density Relationship

Using this density relationship, Singh et al. prove the error performance by comparing the CTL algorithm to a "clairvoyant supervised learner." The clairvoyant supervised learner assumes there exists a pure supervised learning algorithm that has the best possible knowledge of the data. This clairvoyant learner is bounded by the underlying unavoidable error in the dataset, i.e. it cannot be improved.

$$\begin{aligned} Error[\hat{f_{sup,n}}] &\leq \epsilon_{\text{unavoidable}}, \text{ where} \\ \hat{f_{sup,n}} &:= \text{ Clairvoyant learner over n instances} \\ \epsilon_{\text{unavoidable}} &:= \text{Unavoidable error} \end{aligned}$$

The authors then argue if a clairvoyant learner exists, then there is also a clairvoyant CTL SSL classifier for each set. This clairvoyant CTL SSL classifier uses the same algorithm as the pure supervisory classifier, but instead of classifying instances for the entire dataset, the clairvoyant CTL SSL classifier only maps instances for its own cluster. The error bound for the clairvoyant CTL SSL classifier is therefore the same as the pure supervisory classifier within that decision set.

$$\begin{aligned} \forall x \in X; Error[\hat{f_{x,m,n}}] = Error[\hat{f_{sup,n}}] \leq \epsilon_x \leq \epsilon_{\text{unavoidable}}, \text{ where} \\ x &:= \text{a cluster} \\ \hat{f_{x,m,n}} &:= \text{ Clairvoyant CTL learner over m unlabeled, n labeled data in x} \\ \epsilon_x &:= \text{Error in cluster x} \\ X &:= \text{the set of clusters} \end{aligned}$$

Singh et al. complete the proof by placing an upper bound on the accuracy of any CTL SSL classifier by showing how it is equivalent to the error of the clairvoyant CTL SSL classifier plus the potential error due to clustering mistake.

$$\begin{aligned} f_{x,m,n} &\leq \hat{f_{x,m,n}} + O\left(\frac{1}{m} + n\frac{m}{\log^2 m}\right)^{-\frac{1}{d}} \\ &\leq \epsilon_{\text{unavoidable}} + O\left(\frac{1}{m} + n\frac{m}{\log^2 m}\right)^{-\frac{1}{d}}, \text{ where} \\ f_{x,m,n} &:= \text{CTL learner over m unlabeled, n labeled data in x} \end{aligned}$$

The immediate conclusion is that as more unlabeled data, *m*, becomes available, the probability of clustering errors decreases, hence the error due to cluster mistakes decreases. For problem domains with discernable boundaries, this means that the CTL SSL classifier can approach theoretical limits of accuracy with more unlabeled instances. Finally, Singh et al, also demonstrated that while certain cases a CTL approach cannot improve performance, using a CTL does not provable degrade performance either [16].

Practical Issues

Despite this optimistic implication, there are a number of practical issues arriving from the fundamental assumptions of the CTL model. First, this CTL approach assumes "good" clustering behavior, where the unsupervised method used conforms to the discernable decision set margin assumption. Unfortunately, it is far from obvious whether the unsupervised method will conform to this assumption. As [27] demonstrated, a poor choice of clustering can result in the following degenerative situation. In Figure 9, there are four clusters that naturally exist in a population. As Figure 10 demonstrates, the choice of clustering method uses the CTL algorithm to mistake the true decision sets and then trains on the incorrect set of labeled instances. Here complete linkage clustering erroneously links two disjoined sets. The resultant decision set will then train on inaccurate data.

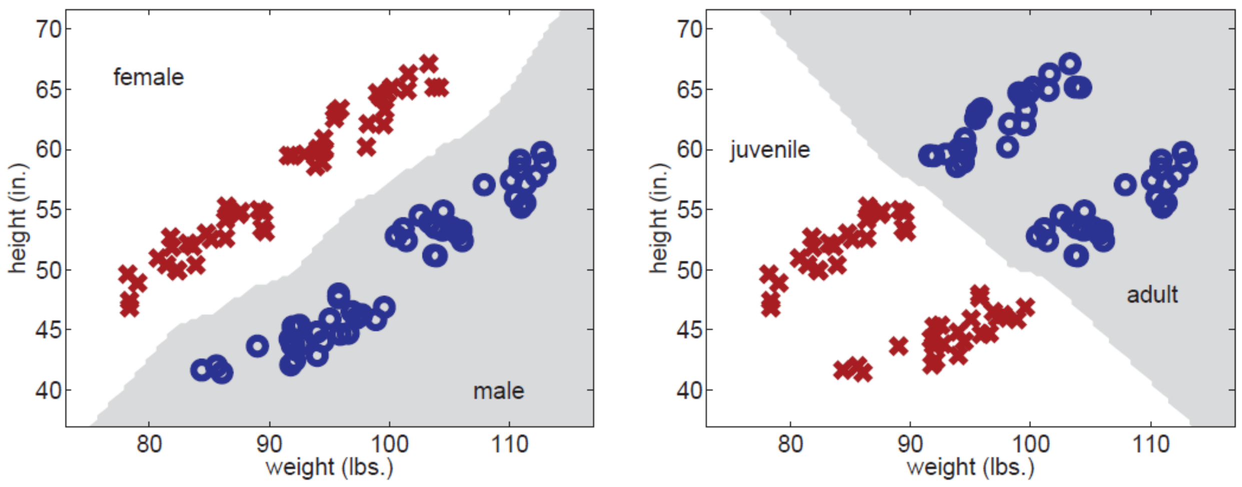


Figure 9: An Example Population Consisting of Four Clusters (*Adapted from* [25])

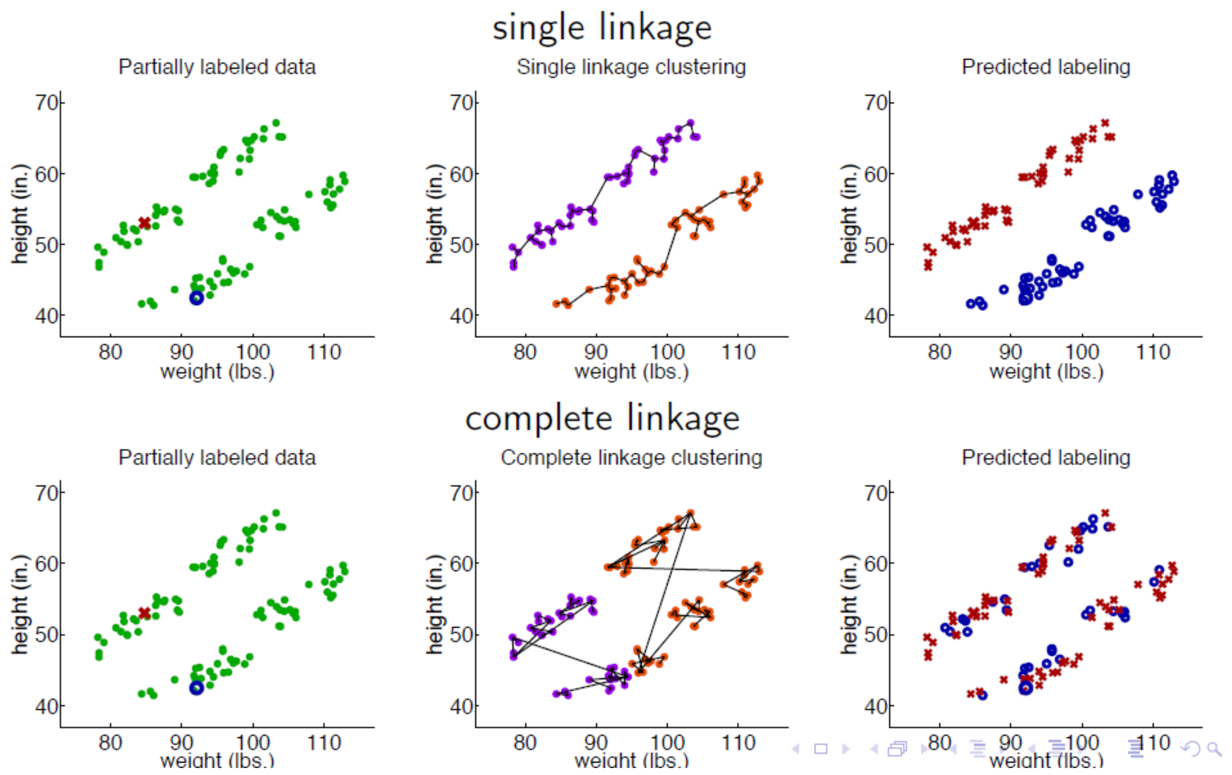


Figure 10: Hierarchical Clustering in Example Population (*Adapted from* [27])

The second issue is that the CTL algorithm assumes a "good" labeling distribution. Typically, any supervised learning algorithm will require that the training dataset is independent and identically distributed (i.i.d.) from the underlying population. While random sampling can ensure that samples are drawn i.i.d. from a population, there is no guarantee that the CTL algorithm will have enough labeled instances per cluster to accurately train each decision set classifier. In a population with features that follow heavy tailed distributions, there may exist clusters with little or no labeled data. Furthermore, certain classes in a population exist at any extreme disproportion to the majority class. Thus, any sampling method, if not comprehensive enough, may not fully capture the minority class of interest. As Chapter 6 will demonstrate, this issue was encountered during the evaluation. One of the promises of SSL is that less labeled information needs to be collected, but it is unclear exactly how much less is needed. With these issues in mind, a researcher may not be able to tell a prior if a dataset has enough labeled data, thus practical implementation may be forced to overestimate the volume of labeled traffic. This sobering reality limits how much effort an SSL approach saves over a pure supervisory approach in a real world setting.

Chapter 4: Dataset

Introduction

This chapter describes the intent and methodology used to create a labeled intrusion detection dataset. First, the "Paradigm Development" section describes how this study created representative samples of stealth scans and the procedure by which those samples were vetted. This chapter also explains the network set-up, implementation choices, and auditing procedures in an effort to be as transparent as possible. This transparency is necessary, because the way the dataset was built could potentially influence the results of any study that uses it.

Paradigm Development

While it is common knowledge that advanced attackers will perform "slow-and-low" stealthy scans, there is no authoritative source that exactly prescribes the perimeters by which attackers scan. To develop a set of stealth scan profiles, I worked with a small group of four students who are members of the University of Maryland Cyber Security Club. All have experience in ethical hacking and penetration testing; two of the students have professional experience. Over a period of five weeks, we trained on the open-source scanning tool, nmap, and dissected how to best scan a network without being detected by traditional means. Nmap is the most popular scanning tool in the security community, and it allows an unparalleled amount of control in setting parameters for scans.

The profiles were developed with intent to determine how a well-resourced attacker would scan a network given a specific objective, a set of exploits to common services and an operational window of no longer than month. The one-month window was chosen because it corresponds to the time between major patch releases. Thus, an attacker could have an entire month to scan a target before it changes. Using nmap, these profiles were tested against an open-source instance of an IDS, Snort. Snort was set with the most stringent port scanning thresholds in order to ensure that the attack would be representative of a stealth scan. In addition, small sets of the profiles were tested against the University's commercial intrusion prevention system (IPS).

Not surprisingly, the scan profiles did not have to deviate significantly from default nmap settings to evade the IDS/IPS. We investigated the two broad categorizes of port scanning based on the scan's intended target footprint: horizontal scans and vertical scans. The main parameters of interest were the target ports and inter-packet timing. We determined that it is possible to infer with good confidence detailed host information while sending a minimal amount of port probes over a long time interval. Table 3 shows the tradeoff between scanning only the most common ports and the percentage of open services discovered. Although the expected percentage of discovered open services may not equal 100%, the scans we developed have enough information to start planning how to exploit the services that are available. Table 4 shows typical settings that were used to evade IDS detection. Additionally, data padding was used in some cases to further obscure detection.

Table 3: Percentage of Services Discovered (*adapted from* [4])

Expected discovery of open services	k most popular TCP Ports	k most popular UDP Ports
10%	1	5
20%	2	12
30%	4	27
40%	6	135
50%	10	1075
60%	18	2618
70%	44	5157
80%	122	7981
85%	236	9623
90%	576	11307
95%	1558	13035
99%	3328	15094
100%	65536	65536

Table 4: Typical Port Scan Settings

	Typical Port Settings	Typical Timing Settings
Vertical Scans	Top port (80), top 10 ports, top 20 ports, top 100 ports, all well-known ports	Min delay between packet {300s, 350s, 400s}, max delay between packet {400s, 500s, 600s}
Horizontal Scans	Top port (80), top 3 ports, top 10 ports, top 20 ports, top 100 ports	Min delay between packet {300s, 350s, 400s}, max delay between packet {400s, 500s, 600s}

In addition to vertical and horizontal scans, we investigated more advanced forms of scanning. We included some instances of coordinated scans, where multiple scanners scan a set of targets. In this paradigm, each scanner IP had a portion of the overall target network's IP addresses and ports. This diffusion makes it harder for a human analyst to see the typical attack scheme in network logs. We also investigated idle or “zombie” scans, where attackers use internal hosts to the target network. In this case, we had trouble reliably scanning the target networks. The limited literature on idle scanning suggests that this experience is not uncommon. The idle scan model requires zombie hosts to have a quiet traffic profile, an obsolete TCP/IP stack implementation, and reliable uptime [28]. Consequently, we did not include idle scanning in this study.

Network Configuration

With profiles developed, we scripted a network of 32 virtual machines (VMs) to scan two Class C subnets of a real production network and instrumented it to collect network flow records, pcap and Snort alerts (see Figure 11 below). Although the experiment focused on network flow records, the other sources of information provided additional insight in case some inconsistency was found. These VMs created two datasets from November through December 2013.

Figure 11: Network Setup

Table 5 5, 6, and 7 summarize the characteristics of this dataset database.

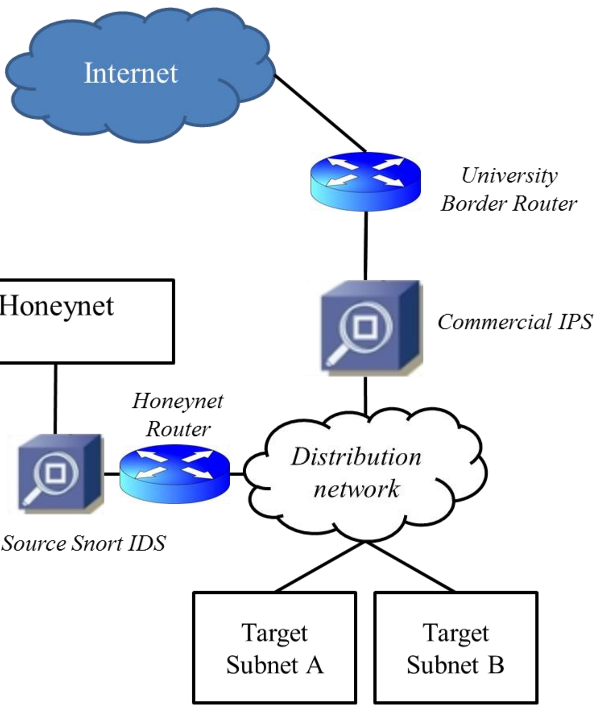


Figure 11: Network Setup

Table 5: November Dataset Characteristics

Duration	30	
Total Number of Raw Flows	12,986,232	97.367%
Total Number of Injected Vertical Scan Flows	4,868	0.036%
Total Number of Injected Horizontal Scan Flows	346,331	2.597%
Total Number of Flows	13,337,431	100.000%
Total Number of Raw Records	2,340,394	98.782%
Total Number of Injected Vertical Records	4,308	0.182%
Total Number of Injected Horizontal Records	24,559	1.037%
Total Number of "Noise records"	29,323	1.238%
Total Number of Adjusted Attack Records	58,190	2.456%
Total Number of Adjusted Raw Records	2,311,073	97.544%
Total Number of Records	2,369,263	100.000%

Table 6: December Dataset Characteristics

Duration	9 Days	
Total Number of Raw Flows	5,025,058	93.143%
Total Number of Injected Vertical Scan Flows	35,349	0.655%
Total Number of Injected Horizontal Scan Flows	334,581	6.202%
Total Number of Flows	5,394,988	100.000%
Total Number of Raw Records	1,004,382	95.600%
Total Number of Injected Vertical Records	24,093	2.293%
Total Number of Injected Horizontal Records	22,134	2.107%
Total Number of "Noise records"	17,792	1.693%
Total Number of Adjusted Attack Records	64,019	6.094%
Total Number of Adjusted Raw Records	986,590	93.906%
Total Number of Records	1,050,609	100.000%

Table 7: January Dataset Characteristics

Duration	28 Days	
Total Number of Raw Flows	24,839,005	98.410%
Total Number of Injected Vertical Scan Flows	19,085	0.076%
Total Number of Injected Horizontal Scan Flows	382,349	1.515%
Total Number of Flows	25,240,439	100.000%
Total Number of Raw Records	1,672,186	98.362%
Total Number of Injected Vertical Records	14,384	0.846%
Total Number of Injected Horizontal Records	13,471	0.792%
Total Number of "Noise records"	35,732	2.102%
Total Number of Adjusted Attack Records	63,587	3.740%
Total Number of Adjusted Raw Records	1,636,454	96.260%
Total Number of Records	1,700,041	100.000%

Data Preprocessing

Feature Set

The University of Minnesota (UMN) Minnesota Intrusion Detection System (MINDS) method incorporates considerable expert knowledge as well as domain-specific knowledge in producing the processed dataset. Tables 8, 9, 10 and 11 enumerate the features that were calculated for this experiment, which was adapted from the UMN MINDS method. The evaluation used these records, which consisted of a key (tuple of source ip, source port and destination port), plus the encoded feature set.

Table 8: Record Key

srcip	External IP, part of key
srcport	Source port of external host, 0 for multiple source ports, part of key
dstport	Destination port port of the internal host, part of key

Table 9: Features of Basic Flow Characteristics

Feature	Description
ndstip	Number of distinct internal IPs touched by the srcip
ndstports	Number of distinct interal ports touched by the srcip
avgdstips	Number of distinct internal IPs averaged over all destination ports touched by the srcip.
maxdstips	Maximum number of distinct internal IPs over all destination ports that the srcip touched

Table 10: Features over All Destination Ports

Feature	Description
server ratio	Ratio of distinct internal IPs that provided the service that the srcip requested to ndstip.
client ratio	The ratio of distinct internal IPs that requested service from the external IP to ndstip.
nosrv ratio	The ratio of distinct internal IPs touched by the srcip that offered no service on dstport to any source during the observation period to ndstip.
dark ratio	The ratio of distinct internal IPs that has been inactive during the experiment window to ndstip.
blk ratio	The ratio of distinct internal IPs that were attempted connections to by the scrcip on a blocked port during the experiment to ndstip.
p2p ratio	The ratio of distinct internal IPs that have actively participated in P2P traffic during the experiment window.

Table 11: Feature on Individual Destination Ports

Feature	Description
i_ndstips	Number of distinct internal IPs touched by the srcip and specific dstport
i_none ratio	Ratio of distinct internal IPs touched by the srcip and specific dstport that did not offer the service requested to i_ndstips
i_dark ratio	Ratio of distinct internal IPs touched by the srcip and specific dstport that did not were not active to i_ndstips
i_blk ratio	Ratio of distinct IPs touched by the srcip to a specific specific blocked dstport to i_ndstips

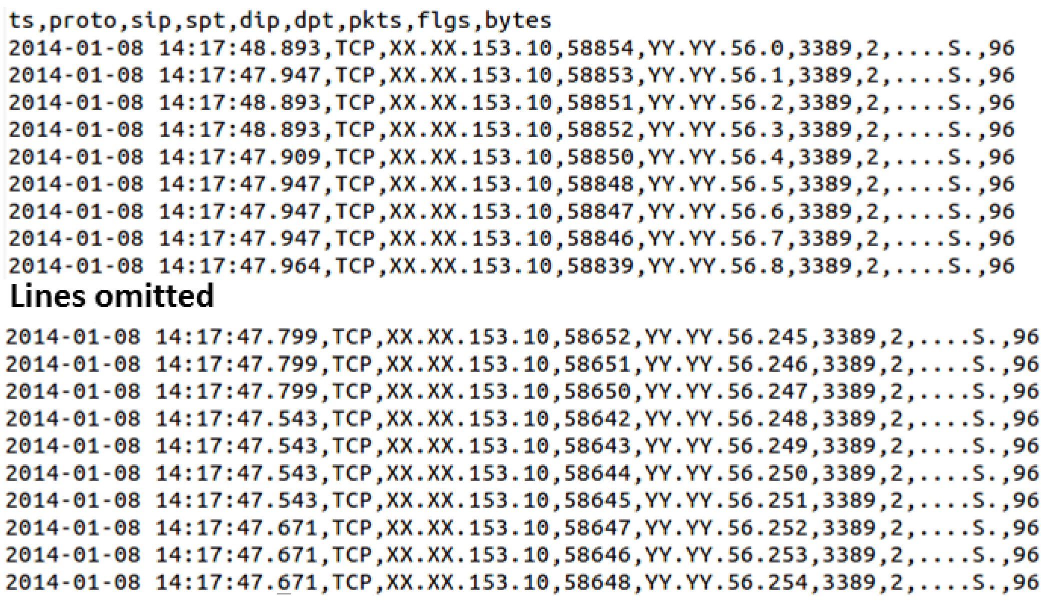
Two implementation choices in the encoding of the feature set above could have had potential impact on this study. First, the UMN method has an implicit encoding of time. The choice is up to the implementer when to calculate the statistics and how far back to calculate the statistics, based on the availability of storage and computation resources. Their implementation calculates the record every 20 minutes for up to a period of 3 days. The size of their defended network limited how many days the UMN researchers could reasonable process in a given time. As Simon et al. suggest, the more days evaluated, the more accurate the method will be [20]. This thesis evaluates the performance of the detection scheme using an evaluation window of 30 days, given the nature of the types of scans evaluated. This choice seems reasonable given the advances in storage and processing big datasets; however, a practical implementation may need a significant amount of optimizations if the network is large.

Second, since every record key consists of a source IP address, there was a potential for the VMs, which were constantly scanning, to have exaggerated statistics. As a result, each complete scan set from the 32 VMs was relabeled with a unique source IP, so that every scan attack seemed to come from a new IP. This choice may be slightly artificial, since in the real world, there is potential for IPs to repeat scans on the same subnet. However, this choice does represent an attack that is harder to distinguish than a scan attack that repeatedly targets the same network and has large statistics on the feature set above.

Data Auditing

All network flow records and processed records were entered into a PostgreSQL database, which enabled manual verification. Network flow records and processed records were stored in separate tables based on whether they were from the network of honeypots or the production network. All records from the network of honeypots were labeled “scanner;” all records from the production network were initially labeled “non-scanner” and then scrutinized for error. In particular, several different queries were used to better check for labeling errors. These queries included manually verifying the most frequent external IP addresses, checking external hosts that initiated flows with a ndstip feature greater than the number of active internal hosts during evaluation period, and checking external hosts that communicated to many well-known ports on the same internal host. By combining the processed record with a filtered network flow table, many scans that were previously unlabeled became evident in the production set. In Table 5, 6 and 7, the noise records are those records that were relabeled. Notice that in November, the amount of noise records actually exceeds the injected traffic.

Figure 12 shows an example of a scan that the queries above catch. By some reorganization of the flow records, it is clear there is horizontal scan from one source IP to an entire subnet looking for hosts that have Remote Desktop Protocol (RDP) services running. RDP is an important Microsoft protocol that allows users to remotely access and control their PCs, and it is a popular target for malicious attackers that are seeking to gain quick control of a poorly administered computer. Records for the scanner IP address (before it was encrypted) reveal that it belongs to one of the major ISPs on the East Coast and is most likely a home user. Based off this information, there is no logical explanation why one home user would be seeking that many RDP connections other than having malicious intent.



```
ts,proto,sip,spt,dip,dpt,pkts,flgs,bytes
2014-01-08 14:17:48.893,TCP,XX.XX.153.10,58854,YY.YY.56.0,3389,2,....S.,96
2014-01-08 14:17:47.947,TCP,XX.XX.153.10,58853,YY.YY.56.1,3389,2,....S.,96
2014-01-08 14:17:48.893,TCP,XX.XX.153.10,58851,YY.YY.56.2,3389,2,....S.,96
2014-01-08 14:17:48.893,TCP,XX.XX.153.10,58852,YY.YY.56.3,3389,2,....S.,96
2014-01-08 14:17:47.909,TCP,XX.XX.153.10,58850,YY.YY.56.4,3389,2,....S.,96
2014-01-08 14:17:47.947,TCP,XX.XX.153.10,58848,YY.YY.56.5,3389,2,....S.,96
2014-01-08 14:17:47.947,TCP,XX.XX.153.10,58847,YY.YY.56.6,3389,2,....S.,96
2014-01-08 14:17:47.947,TCP,XX.XX.153.10,58846,YY.YY.56.7,3389,2,....S.,96
2014-01-08 14:17:47.964,TCP,XX.XX.153.10,58839,YY.YY.56.8,3389,2,....S.,96
Lines omitted
2014-01-08 14:17:47.799,TCP,XX.XX.153.10,58652,YY.YY.56.245,3389,2,....S.,96
2014-01-08 14:17:47.799,TCP,XX.XX.153.10,58651,YY.YY.56.246,3389,2,....S.,96
2014-01-08 14:17:47.799,TCP,XX.XX.153.10,58650,YY.YY.56.247,3389,2,....S.,96
2014-01-08 14:17:47.543,TCP,XX.XX.153.10,58642,YY.YY.56.248,3389,2,....S.,96
2014-01-08 14:17:47.543,TCP,XX.XX.153.10,58643,YY.YY.56.249,3389,2,....S.,96
2014-01-08 14:17:47.543,TCP,XX.XX.153.10,58644,YY.YY.56.250,3389,2,....S.,96
2014-01-08 14:17:47.543,TCP,XX.XX.153.10,58645,YY.YY.56.251,3389,2,....S.,96
2014-01-08 14:17:47.671,TCP,XX.XX.153.10,58647,YY.YY.56.252,3389,2,....S.,96
2014-01-08 14:17:47.671,TCP,XX.XX.153.10,58646,YY.YY.56.253,3389,2,....S.,96
2014-01-08 14:17:47.671,TCP,XX.XX.153.10,58648,YY.YY.56.254,3389,2,....S.,96
```

Figure 12 Embedded Scan (source and target IP obscurred and masked)

Chapter 5: Experiment

Experimental Parameters

This thesis used a series of trials to evaluate the performance of the UMN MINDS algorithm as well as the CTL algorithm. The evaluated implementation of CTL used k-means++ with simple Euclidian distance and RIPPER for the supervised classifier with two rounds of optimization. Two trials were executed using only the control algorithm: one with the network of honeypots traffic naively injected and one with records adjusted. These trials demonstrate the effect that noise may have when building a classifier with injected attack data. Next, the datasets were broken into sets of randomly selected labeled instances mixed with unlabeled instances. The motivation behind creating these sets is to compare the CTL algorithm performance against a pure supervisor and to illustrate how much labeled data would be needed to achieve acceptable levels of accuracy. Trial I attempts to show how a practical approach a security administrator might take to labeling a dataset, for instance from a company network. In this setting, the administrator combines penetration testing data with the existing logs. The administrator would have audited only the most frequent hosts, which he/she would label as scanner or non-scanner as appropriate. In addition to varying the proportions of labeled traffic, we varied the number of clusters across all CTL implementations

Tables 12, 13 and 14 capture the the experimental parameters that were used in each trial for the test and control cases.

Table 12: Experimental Parameters over Supervised Trial Cases

Trial	Training set	Test set	Number of labeled Instances	Number of labeled attack Instances	Number of distinct Ips	Total Size of Training set
A	NOV; fully labeled	10-CV; Adjusted NOV full	2,369,263	28,867	287,974	2,369,263
B	Adjusted NOV; fully labeled	10-CV	2,369,263	58,190	287,974	2,369,263

Table 13: Experimental Parameters over CTL Trial Cases

Trial	Training set	Test set	Number of clusters evaluated	Number of labeled Instances	Number of labeled attack Instances	Number of distinct Ips	Total Size of Training set
C	75% randomly selected labeled instances, mixed from NOV	DEC, fully labeled; JAN, fully labeled	{2-90}	1,781,360	43,700	234,221	2,369,263
D	50% randomly selected labeled instances, mixed from NOV	DEC, fully labeled; JAN, fully labeled	{2-90}	1,194,064	29,479	175,645	2,369,263
E	25% randomly selected labeled instances, mixed from NOV	DEC, fully labeled; JAN, fully labeled	{2-90}	607,608	15,160	108,284	2,369,263
F	10% randomly selected labeled instances, mixed from NOV	DEC, fully labeled; JAN, fully labeled	{2-90}	254,952	6,352	57,895	2,369,263
G	5% randomly selected labeled instances, mixed from NOV	DEC, fully labeled; JAN, fully labeled		137,755	3,515	36,693	2,369,263
H	1% randomly selected labeled instances, mixed from NOV	DEC, fully labeled; JAN, fully labeled	{2-90}	43,183	1,228	15,333	2,369,263
I	top 30 benign, top 5 scanners, mixed from NOV	DEC, fully labeled; JAN, fully labeled	{2-100}	302,300	32,303	35	2,369,263
J	1% randomly selected labeled instances plus injected labeled scan traffic, mixed from NOV	DEC, fully labeled; JAN, fully labeled	{2-90}	82,718	40,764	15,356	2,369,263

Table 14: Experimental Parameters over CTL Control Cases

Trial	Training set	Test set	Number of clusters evaluated	Number of labeled Instances	Number of labeled attack Instances	Number of distinct Ips	Total Size of Training set
C_control	75% randomly selected only labeled instances from NOV	DEC, fully labeled; JAN, fully labeled	N/A	1,781,360	43,700	234,221	1,781,360
D_control	50% randomly selected only labeled instances from NOV	DEC, fully labeled; JAN, fully labeled	N/A	1,194,064	29,479	175,645	1,194,064
E_control	25% randomly selected only labeled instances from NOV	DEC, fully labeled; JAN, fully labeled	N/A	607,608	15,160	108,284	607,608
F_control	10% randomly selected labeled instances, mixed from NOV	DEC, fully labeled; JAN, fully labeled	N/A	254,952	6,352	57,895	254,952
G_control	5% randomly selected only labeled instances from NOV	DEC, fully labeled; JAN, fully labeled	N/A	137,755	3,515	36,693	137,755
H_control	1% randomly selected only labeled instances from NOV	DEC, fully labeled; JAN, fully labeled	N/A	43,183	1,228	15,333	43,183
I_control	top 30 benign, top 5 scanners, mixed from NOV	DEC, fully labeled; JAN, fully labeled	N/A	302,300	32,303	302,300	302,300
J_control	1% randomly selected labeled instances plus injected labeled scan traffic, mixed from NOV	DEC, fully labeled; JAN, fully labeled	N/A	82,718	40,764	15,356	82,718

Each trial was evaluated with the following measures:

$$\begin{aligned}
\text{Accuracy} &= \frac{\text{TP} + \text{TN}}{\text{N}} \\
\text{Precision} &= \frac{\text{TP}}{\text{TP} + \text{FP}} \\
\text{Specificity} &= \frac{\text{TN}}{\text{TN} + \text{FP}} \\
\text{Recall/Sensitivity} &= \frac{\text{TP}}{\text{TP} + \text{FN}} \\
\text{Fm} &= \frac{2 \cdot \text{Prec} \cdot \text{Recall}}{\text{Prec} + \text{Recall}}
\end{aligned}$$

Figure 13: Performance Measures

The true positive (TP), true negative (TN), false positive (FP), and false negative (FN) are given by Table 15. Recall, also known as sensitivity, corresponds to the relative frequency of correctly classified scanner instances. Precision captures the proportion of correctly classified scanner instances. The F-measure captures the balance between recall and precision, so that an ideal classifier that has a low rate of false positives and false negatives will achieve a F-measure close to one. It is worth noting that the ROC score (sensitivity versus false positive rate), which is popular measure, was not calculated. We did not use the ROC score because the balance between the true positive rate and false positive rate was not a monotonic function of some threshold in this experiment. Thus, ROC curves did not add any meaningful insight into a classifier’s performance beyond what was already captured by the metrics above [29].

Table 15: Classification Matrix (*Adapted from* [20])

	Classified As Scanner	Classified as Not Scanner
Actual Scanner	TP	FN
Actual Non-scanner	FP	TN

McNemar's test was used for testing statistical significance [30]. Unless explicitly stated below, all trials in this test showed statistical distinction from their control, with the probability of Type I error < 0.5% .

<u>*CTL Implementation*</u>

There is no off-the-shelf program with CTL already implemented. I implemented the main portions of CTL in Java, which enabled the use of the Weka machine learning library. Weka has implementations of RIPPER and k-means++, and a well-documented API.

Tests were executed on a 64-bit virtual machine with 16 GB of RAM, 8 x 2643 MHz CPUs.

Chapter 6: Results and Discussion

The first two trials demonstrate that naively injected traffic into a dataset without verification can have disastrous results. In Table 16, the first iteration of Trial A using the unrectified NOV dataset showed high accuracy in a 10 fold cross-validation test. Using Trial A's classifier, it was then evaluated against the adjusted NOV dataset, where it produced 29,317 more false negatives. RIPPER is theoretically tolerant of noise, but this property has limits. If the amount of training data that is mislabeled as “normal” is on the order of the injected attack traffic, then the algorithm will overfit to the injected traffic.

Table 16: Supervised Trials

	Spec	Acc	Precision	Recall/sens	Fm
Trial A, 10-CV	0.999998	0.999990	0.999861	0.999307	0.999584
Trial A, Adjusted NOV full	0.999998	0.987616	0.999861	0.495807	0.662898
Trial B	1.000000	0.999992	1.000000	0.999656	0.999828

The general trend in the results from trials C-J is that the CTL algorithm is more sensitive to the choice in number of clusters than the theoretical background would suggest (see Figure 14). Trials C-H and J in particular exhibited this behavior (Figures 14, 15, 16, and 18). The CTL implementation consistently failed to outperform the control when the number of clusters was not ideal, and in some cases the CTL performance was marginally poorer. For most trials, the ideal number of clusters is around 50. In this range (30 to 70), the CTL method will typically outperform the pure supervisory algorithm in all measures. For trials C-H and J this performance increase was marginal compared to the control; for trial I (Figure 17) the performance increase was significant.

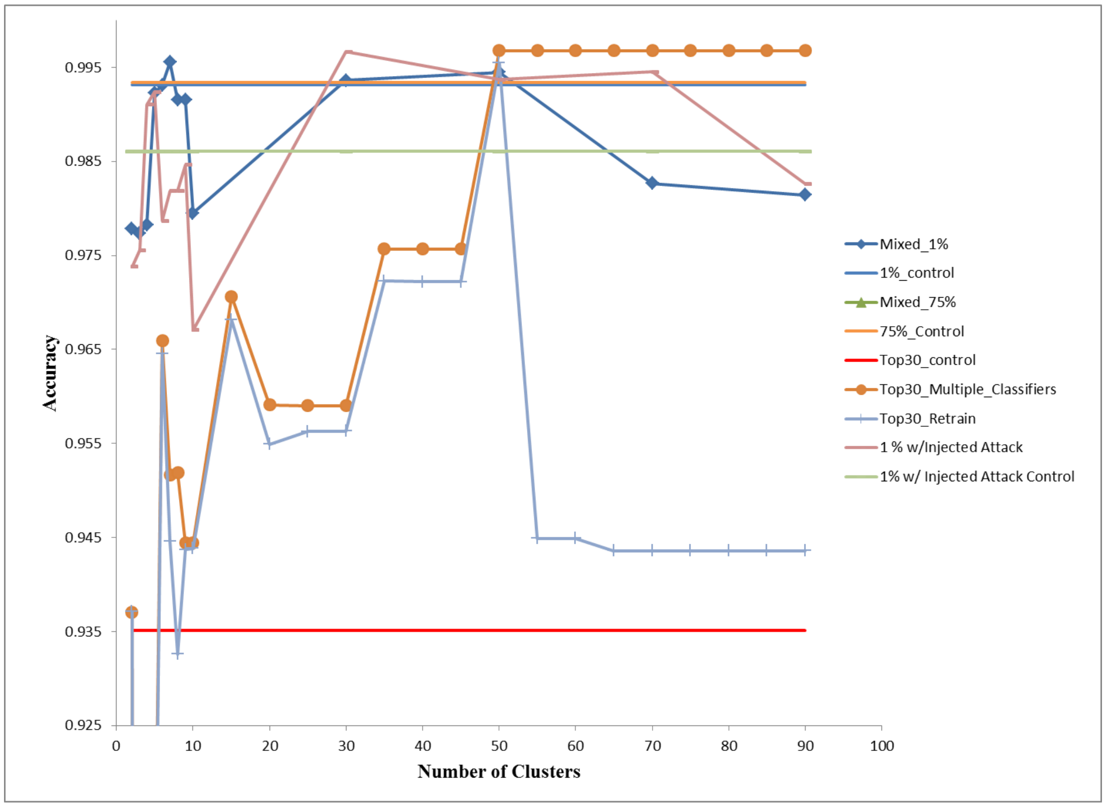


Figure 14: Performance of CTL Classifiers with Different Amounts of Labeled Data

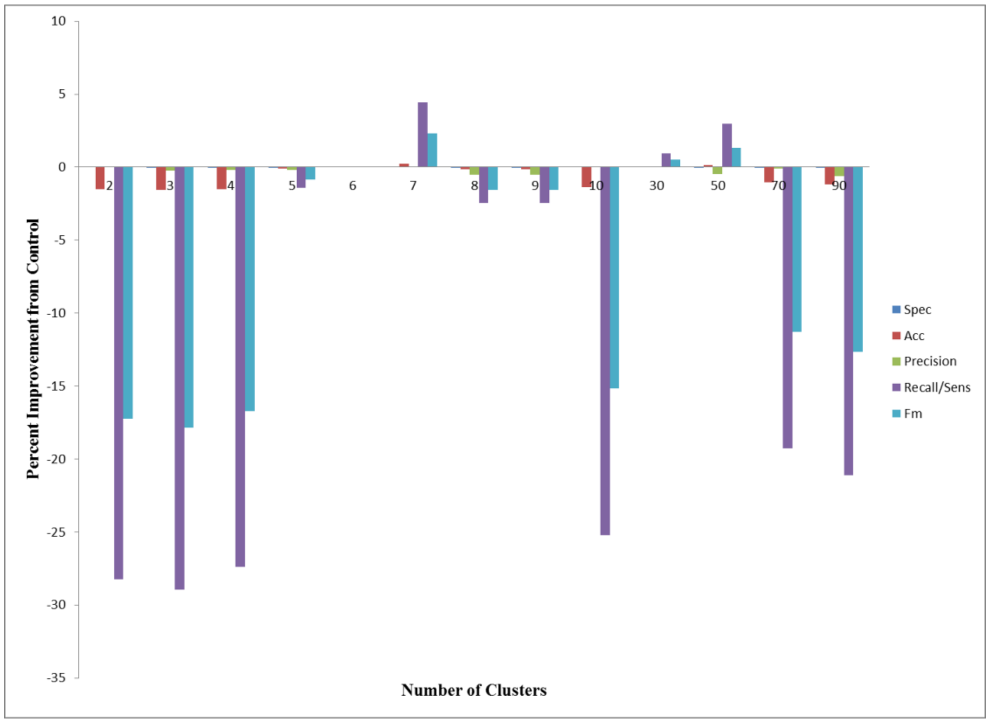


Figure 15: Improvement from December Trial H (1 % Labeled)

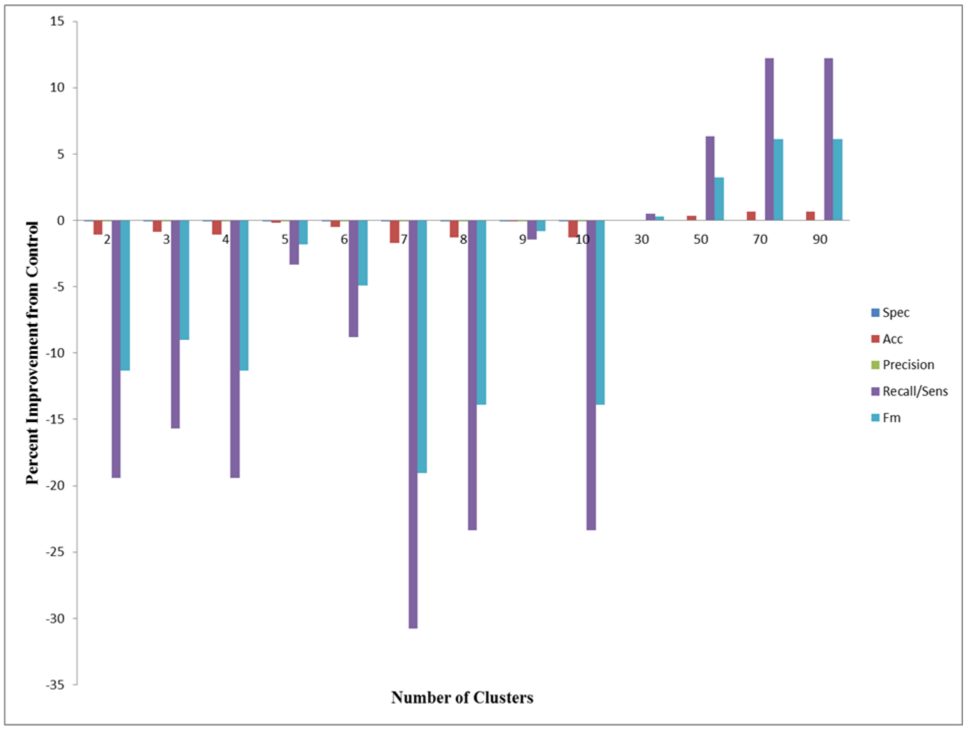


Figure 16: Improvement from December Trial C(75% labeled)

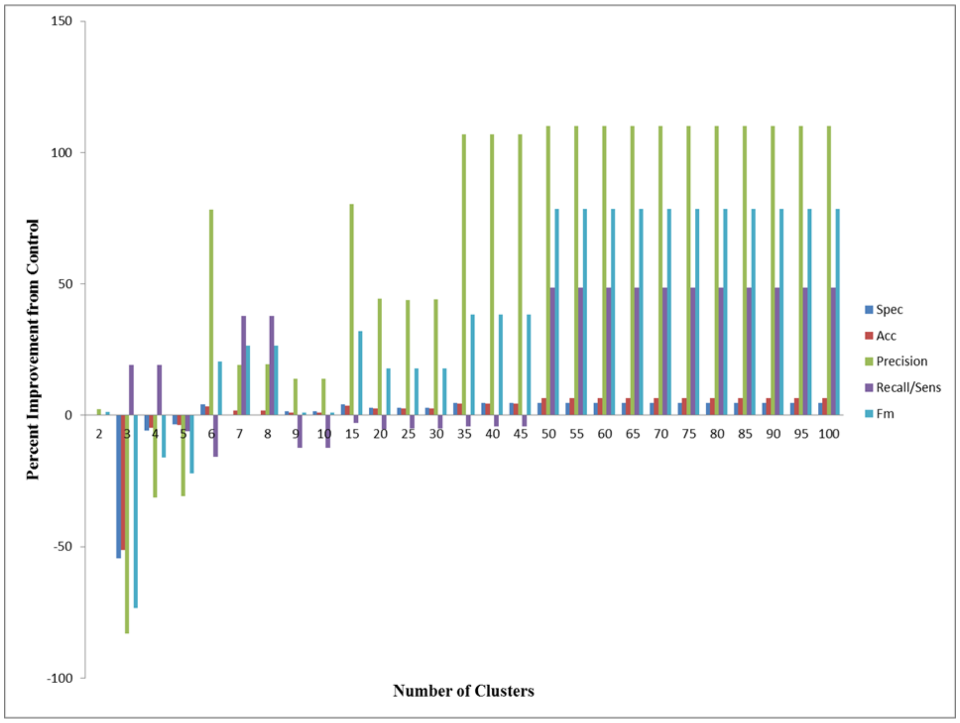


Figure 17: Improvement from December Trial I (top 30 non-scanner, top 5 scanners labeled)

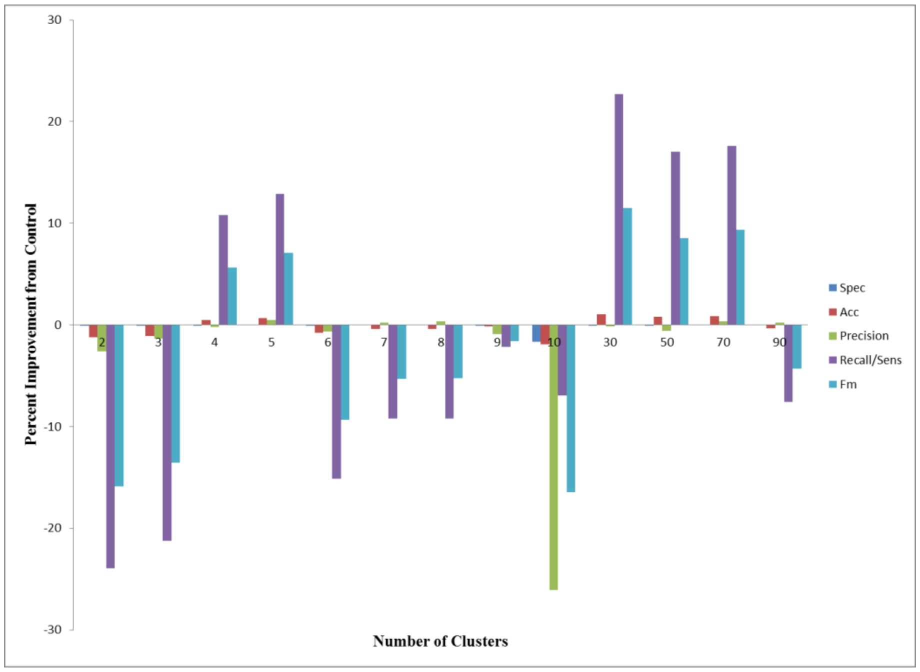


Figure 18: Improvement from December Trial J (1 % random with injected attack traffic labeled)

When the cluster assignment is not ideal, the CTL algorithm typically has poor sensitivity, therefore a high false negative rate, compared to the control. The high false negative rate is primarily due to inherent disproportion between scan and non-scan classes. Even though the sampling method conformed to the i.i.d. imperative, because such a small percentage of scan traffic is present in the dataset, the sampling procedure did not capture enough instances to accurately train each decision set classifier. To illustrate this phenomenon, Figure 19, 20, and 21 show the number of instances of each class label from a 1% randomly selected training set and from a full month test set.

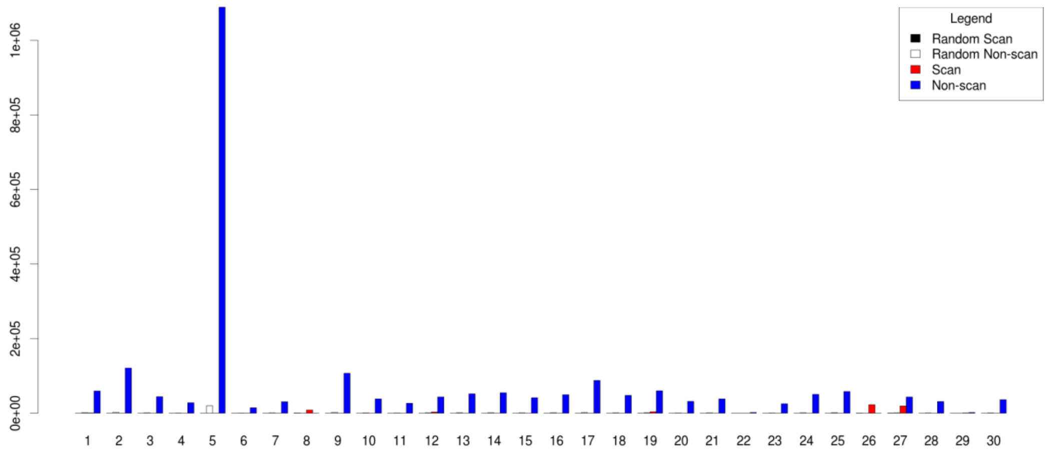


Figure 19: Proportions of Training and Test Set Traffic per Cluster

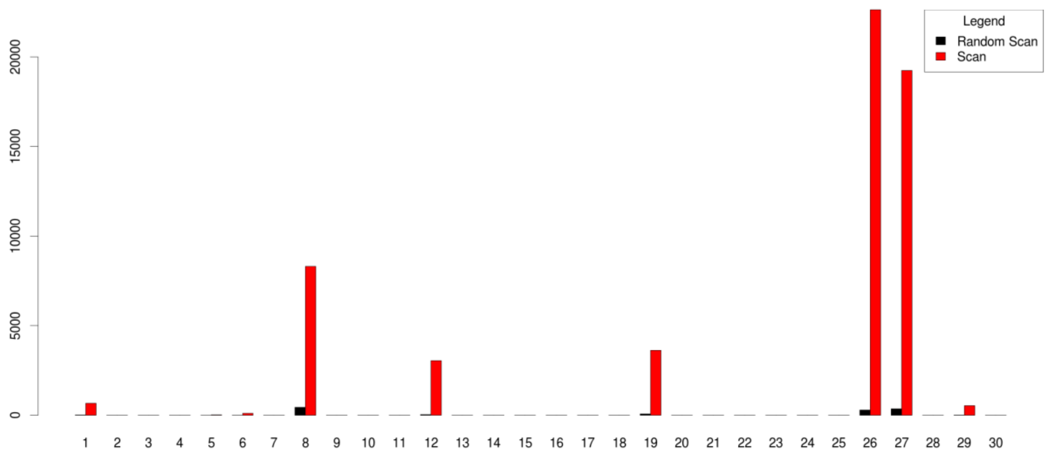


Figure 20: Proportions of Training and Test Set Scan Traffic per cluster

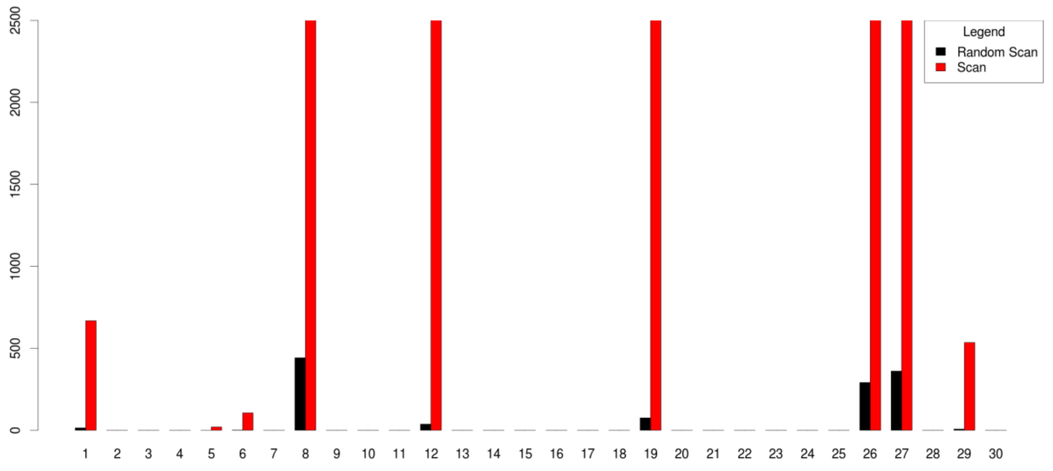


Figure 21: Proportions of Training and Test Set Scan Traffic per Cluster (Scaled Between 0 To 2500)

It becomes clear in Figure 21 that there are certain clusters in the training set that never received a sufficient amount of labeled scan traffic for RIPPER to work properly. Table 17 shows that there are a few clusters that experience little to no scans in the training set but experience a high number of scans in the test set. Since RIPPER is seeking to separate out the minority (scan) class on its distinguishing features, if RIPPER does not train on enough representative minority class samples, the algorithm will produce a large amount of false negatives. While it is obvious that any supervised algorithm performs better with more training data, since the proportion of normal traffic is much greater than the proportion of attack traffic, simple sampling is not enough. In situations where the number of clusters is small, it may actually be better to inject more attack traffic than what occurs at natural proportions. This skewing of the data set helps to induce the proper bias for the classifier. Finally, if there are too many clusters, the false negative rate also becomes high. This problem is caused by the CTL algorithm subdividing an ideal decision cluster into multiple subpar clusters. Since the subpar clusters have little labeled instances, their internal cluster classifier may default to majority voting. Any future instance will be labeled with the majority class label (i.e. non-scanner); therefore, subdividing an ideal decision cluster may result in unlabeled scanner instances being improperly classified as non-scanner.

Table 17: Number of Scans Per Cluster in Test and Training Sets

Cluster	Number of labeled Scan instances in the training set	Number of Scan instances in the test set
1	14	668
2	0	0
3	0	0
4	0	0
5	0	20
6	1	106
7	0	0
8	442	8310
9	0	0
10	0	0
11	0	0
12	37	3046
13	0	0
14	0	0
15	0	0
16	0	0
17	0	0
18	0	0
19	75	3623
20	0	0
21	0	0
22	0	0
23	0	0
24	0	0
25	0	0
26	291	22634
27	361	19248
28	0	0
29	7	535
30	0	0

As confirmation of the effects of injecting more scan traffic, trial J (Figure 18) consists of a mixed data set comprised of a random selection of 1% of the labeled NOV traffic and all the labeled traffic from 23 dedicated scanners (both the injected scan records and records that were discovered in the dataset during auditing). While trial J ran with a lower false negative rate, it is still sensitive to the choice of cluster numbers.

Despite only showing marginal performance gains in trial C-H and J, trial I showed more promising results for a real world implementation. Here the performance gains were significant. Before the trial, I conjectured that this choice of the top 30 IP addresses for the "normal" labeled portion may skew the classifier. Most of the IP addresses on the targeted subnet are servers, but in the network flows associated with the top 30 IP addresses, the UMD internal hosts are primarily acting as clients. Surprisingly, the choice of top 30 benign IP addresses does not skew the data much in the CTL approach. While there was no significant impact in this study, this choice may affect another set of malicious activity in an unintended way. However, this technique does offer hope that in a practical implementation a security administrator would only have to audit a small amount of the most frequently seen IP addresses in order to build an adequate classifier.

Interestingly, the supervised classifier in each cluster has fewer rules when the number of clusters increases. For implementations with the largest number of clusters, the decision set supervised classifier is just majority voting, i.e. traffic gets labeled with the label of the class that makes up the majority of that cluster. This behavior reinforces the idea the data for this attack type behaves "nicely" and naturally occurs in close-nit clusters, for the optimal *k*. A practical real world implementation could exploit the natural clustering by calculating a large number of *k* clusters, and then labeling each centroid with the majority class label. Future instances could then be classified using a nearest neighbor search on the cluster centroids.

Some analysis was conducted to assess the impact that domain-specific knowledge has on classification accuracy. Some of the features require extensive encoding of a priori knowledge such as the set of block ports per host. Ideally, smart feature selection could still have good performance without extensive amounts of overhead. While some initial research showed potential, the analysis was halted because of some inherent limitations. First, in the January dataset, there is a spike in the prevalence of cloud services that use protocols communicating on multiple registered ports (1024 < registered ports < 30k). This spike makes it difficult to determine if vertical scans are being performed on certain hosts without logic that checks if that port on a host is open, closed or filtered. Furthermore, it started to become apparent that rules the RIPPER was learning rules that were specific to the evaluation network. For instance, one of the rules RIPPER developed accurately capture the number of distinct IP addresses that were active on the subnet during the evaluation window, but this number may not generalize to another subnet with a different distinct number of IP addresses. While the domain-specific knowledge requires some overhead, the way the statistics are calculated do allow it to generalize better to other subnets.

There are some minor issues that are network specific. On our test network, the network logs time stamps did not always provide the necessary accuracy to state which external IP was a server or a client. The router appears to arbitrarily order complimentary network flows with identical timestamps; therefore, our implementation defaulted to labeling external IP addresses as clients when the time stamps were equal and the external port was outside the well-known and registered range. This way of handling identical time stamps could introduce noise, and the approach described in this thesis could exhibit better performance on more accurate logs. Also, this lack of fidelity weakens distance measures. In every situation, where an external host cannot be deterministically declared as a client or server, the implementation defaults to client. As a result, certain statistics like client ratio, have few nonzero entries. A high occurrence of a default value will make the Euclidean distance between two disparate instances seem smaller than if the distance was calculated with the feature was omitted.

The performance of trial A and B exhibited better performance than what was originally reported in the UMN MINDS study. This difference is most likely due to our study using a 30-day evaluation window. Also, this study used two focused subnets with minimal peer-to-peer traffic as opposed to the UMN study which used records from the entire campus.

Unfortunately, the base-rate fallacy still limits an adoption of this approach. While some iterations did achieve an accuracy that may be acceptable (> 99.5%), most iterations did not and a user of this approach would have to perform extensive evaluation to find the optimal number of clusters. In addition, auditing the existing traffic for the presence of noise to a specific attack type requires sizable analysis. For scanning, it is verbose enough to identify after some initial processing. Other attack paradigms may not be so accommodating. Without this extensive auditing, the accuracy may fall to a level where the base-rate fallacy prevents its use.

Finally, a pleasant surprise came from a mistake in the way one of the datasets was created. When creating the mixed label set for Trial J, the original test did not use the correct file with both labeled and unlabeled instances. The CTL implementation was executed on a dataset set that on only contained labeled data (Figure 22). In this context, this classifier is not acting in as a semi-supervised learner but as a two-stage supervised classifier: clustering on the labeled instances in the first stage and pure supervised learning within the clusters in the second. This two-stage classifier exhibited high performance across all numbers of clusters. While a full analysis of this approach is beyond this thesis, one potential explanation that RIPPER is experiencing a performance gain in choosing how to divide the instances based on the feature values. One way to interpreted RIPPER's output is as a function that maps an instance based on a series of conditional probabilities. RIPPER uses a greedy algorithm to select dividing points in the values of a feature's domain. In most implementations there appears to be no advanced statistics on continuous values, just sorting and selection. When restricted to subsets of the instances, RIPPER may be better able to learn dividing points since there is less likely to be variance and skewness in the range of a feature's values within a cluster. A two-stage classifier offers hope that rule-based and decision tree algorithms can be used in domains like intrusion detection where perfect knowledge of underlying distributions is often infeasible.

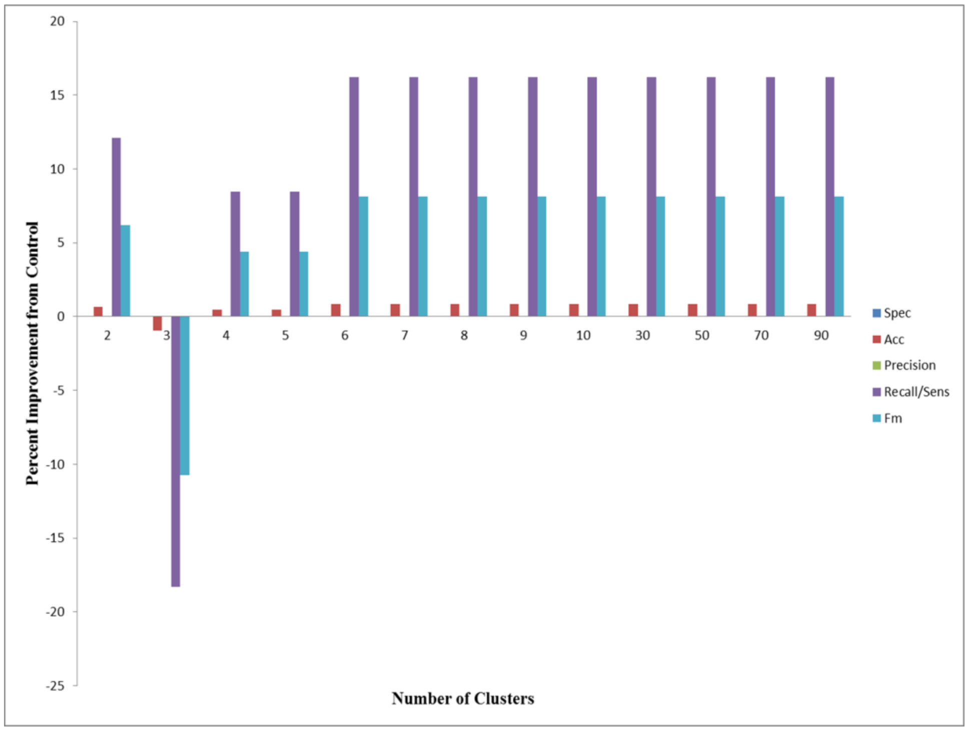


Figure 22: Performance of a Two-stage Classifier

Chapter 7: Conclusions

This thesis illustrated how CTL SSL algorithms can build classifiers with comparable accuracy with a fraction of the labeling effort as traditional supervised learning, provided the number of clusters is ideal. Performance gains vary, but in general the trend is that at the ideal cluster value they will do better than the supervised counterpart. While lending promise to security researchers, enthusiasm for SSL needs to be tempered with consideration for practical issues such as noise in the normal traffic. Poor clustering can actually slightly degrade performance. Careful consideration should also be given for the choice unsupervised and supervised learners and their parameters.

There are numerous avenues for future work. First, a comprehensive evaluation of CTL using other supervised and unsupervised learning methods with different heuristics and distances measures should be performed. The choice of RIPPER was driven by the need for interpretable results, speed, and comparison to existing measures. The performance gain for RIPPER may not be as pronounced as it is for other algorithms. The theoretical analysis suggests that the CTL algorithm should have a markedly better performance than a pure supervised algorithm given enough unlabeled data. That being stated, RIPPER is not as constrained as other approaches, such as SVM. It may stand to reason that an application that lends itself to SVM or other linear discriminative methods may have better performance using a CTL approach.

For unsupervised clustering, the Euclidean distance was the only distance measure evaluated here. The Euclidean distance does not always perform the best.

Certain low variance features, like client ratio, weaken un-weighted measures of distance. Thus, there is potential when using Euclidean distance measures in a set with a large number of features or many similar valued features that all instances may appear close. In this situation, other weighted distances and compensated distance measures can account for features where the domain lacks variance or have missing values. These techniques might be more appropriate here.

While initially a mistake and outside the scope of this thesis, a two-stage approach shows promise. In particular, it may be better able to handle imbalanced datasets or datasets with proportions of classes that deviate from the natural population. A two-stage approach warrants further investigation.

Finally, this approach should be evaluated on other attack paradigms and datasets to see how the traffic characteristics of different attacks affect detection in network flow data. Scanning produces many network flow records; so even before preprocessing, there are many instances within a dataset to train a machine learning algorithm. Other sets of activity may prove too difficult to detect accurately without additional information sources. Related to this study, more research should focus on how evidence of scanning can be used in conjunction with other evidence to detect other sets of activities. While Lane's work in modelling POMDP in order to detect UNIX terminal misuse is a start, a multi-disciplinary approach should cover detection for all know threat frameworks [12]. An advanced hacker conducts scanning in a radically different manner than the way a worm conducts its automated scans, so belief networks should be built according to the know behavior of threats.

References

[1] C. T. Symons and J. M. Beaver, "Nonparametric Semi-Supervised Learning for Network Intrusion Detection: Combining Performance Improvements with Realistic In-Situ Training," *AISec '12 Proceedings of the 5th ACM Workshop on Security and Artificial Intelligence,* pp. 49-58, 2012.

[2] B. Claise, "Cisco Systems NetFlow Services Export Version 9," *IETF, RFC 3954,* October 2004.

[3] Cisco, "Introduction to Cisco IOS NetFlow," May 2012. [Online]. Available: www.cisco.com.

[4] G. Lyon, Nmap Network Scanning: Offical Nmap Project Guide to Network Discovery and Security Scanning, Sunnyvale: Insecure, 2008.

[5] S. Dau and X. Du, Data Mining and Machine Learning in Cybersecurity, Boca Raton: Auerbach, 2011.

[6] R. Sommer and V. Paxson, "Outside the Closed World: On Using Machine Learning For Network Intrusion Detection," in *IEEE Symposium on Security and Privacy*, Oakland, 2010.

[7] S. Axelsson, "The Base-Rate Fallacy and the Difficulty of Intrusion Detection," in *ACM Transactions on Information and System Security (TISSEC)*, 2008.

[8] A. Sperotto, R. Sadre, F. Van Vliet and A. Pras, "A Labeled Data Set For Flow-based Intrusion Detection," in *IP Operations and Management*, Berlin, Springer, 2009, pp. 33-50.

[9] M. Tavallaee, E. Bagheri, W. Lu and A.-A. Ghorbani, "A Detailed Analysis of the KDD CUP 99 Data Set," in *Proceedings of the Second IEEE Symposium on Computational Intelligence for Security and Defence Applications*, 2009.

[10] H. Takakura, Y. Okabe, M. Eto, D. Inoue, K.. Nakao, & J Song, "Statistical analysis of honeypot data and building of Kyoto 2006+ dataset for NIDS evaluation," in *Proceedings of the First Workshop on Building Analysis Datasets and Gathering Experience Returns for Security*, Salzburg, 29-36.

[11] M. Dornseif, T. Holz and C. N. Klein, "NoSEBrEaK--Attacking Honeynets," in *Workshop on Information Assurance and Security*, West Point, 2005.

[12] T. Lane, "A Decision-Theoretic, Semi-Supervised Model for Intrusion Detection," in *Machine Learning and Data Mining for Computer Security*, London, Springer, 2006, pp. 157-177.

[13] C.-H. Mao, H.-M. Lee, D. Parikh, T. Chen and S.-Y. Huang, "Semi-upervised Co-training and Active Learning Based Approach for Multi-View Intrusion Detection," in *Proceedings of the 2009 ACM symposium on Applied Computing*, 2009.

[14] A. B. Goldberg, X. Zhu, A. Singh, Z. Xu, and R.D. Nowak, "Multi-Manifold Semi-Supervised Learning," *International Conference on Artificial Intelligence and Statistics,* pp. 169-176, 2009.

[15] A. Demiriz, K. P. Bennett and M. J. Embrechts, "Semi-supervised clustering using genetic algorithms," in *Artificial Neural Networks in Engineering* , 1999.

[16] A. Singh, R.D. Nowak, and X. Zhu, "Unlabeled data: Now it helps, now it

doesn't," *NIPS,* pp. 1513-1520, 2008.

[17] S. Staniford, J. A. Hoagland and J. M. McAlerney, "Practical Automated Detection of Stealthy Portscans," *Journal of Computer Security,* pp. 105-136, 2002.

[18] J. Jung, V. Paxson, A. W. Berger and H. Balakrishnan, "Fast Portscan Detection using Sequential Hypothesis Testing.," in *Security and Privacy, 2004. Proceedings. 2004 IEEE Symposium on*, 2004.

[19] C. Muelder, K.-L. Ma and T. Bartoletti, "A Visualization Methodology for Characterization of Network Scans," in *IEEE Workshop on Visualization for Computer Security,* 2005.

[20] G. J. Simon, H. Xiong, E. Eilertson and V. Kumar, "Scan Detection: A Data Mining Approach," in *Proceedings of the Sixth SIAM International Conference on Data Mining*, SIAM, 2006.

[21] W. W. Cohen, "Fast Effective Rule Induction," *Machine Learning Proceedings of the Twelfth International Conference,* 1995.

[22] I. Witten, and E. Frank*, Data Mining: Practical Machine Learning Tools and Techniques,* 2nd ed., New York: Elsevier, 2005.

[23] T. Menzies, "Data Mining: Fall '06," 2006. [Online]. Available: http://www.csee.wvu.edu/~timm/cs591o/old/Rules.html. [Accessed January 2014].

[24] D. Arthur and S. Vassilvitskii, "K-means++: The Advantages of Careful Seeding," in *Proceedings of the eighteenth annual ACM-SIAM symposium on*

Discrete algorithms, 2007.

[25] X. Zhu and A. B. Goldberg, Introduction to Semi-Supervised Learning, Madison: Morgan and Claypool, 2009.

[26] X. Hu, M. Knysz, and K. Shin, "RB-Seeker: Auto-detection of Redirection Botnets," *NDSS,* 2009.

[27] X. Zhu, ""Tutorial on Semi-Supervised Learning"," in *Theory and Practice of Computational Learning,* Chicago, 2009.

[28] M. Morbitzer, "TCP Idle Scanning Using Network Printers".

[29] I. Kononenko and M. Kukar, Machine Learning and Data Mining: Introduction to Principles and Algorithms, West Sussex: Horwood, 2007.

[30] T. G. Dietterich, "Approximate Statistical Tests for Comparing Supervised Classification Learning Algorithms," *Neural Computation,* vol. 10, no. 7, pp. 1895-1923, 1998.

[31] C. T. Symons, Interviewee, *Questions on Nonparametric Semi-Supervised Learning for Network Intrusion Detection.* [Interview]. 17 September 2013.

[32] C. Thomas, *Application of Machine Learning for Intrusion Detection: Challenges and Solutions,* IEEE Transactions. Manuscript submitted for publishing., 2013.

[33] J. McHugh, "Testing Intrusion Detection Systems: A Critique of the 1998 and 1999 DARPA Intrusion Detection System Evaluation as Performed by Lincoln Laboratory," *ACM Tranactions on Information System Security,* vol. 3, no. 4, pp. 262-294, 2000.

[34] T. Holz and F. Raynal, "Detecting Honeypots and Other Suspicious Environments," in *Workshop on Information Assurance and Security*, West Point, 2005.

[35] J. Quittek, S. Bryant, B. Claise, P. Aitken and J. Meyer, "Information Model for IP Flow Information Export," *IETF, RFC 5102,* January 2008.

[36] B. Claise, "Specification of the IP Flow Information Export (IPFIX) Protocol," *IEFT, RFC 5101*, January 2008.

[37] T. Mitchell, Machine Learning, Boston: McGraw-Hill, 1997.