HANDLING UNCERTAINTY IN INTRUSION ANALYSIS

by

LOAI M. M. ZOMLOT

B.S., Islamic University, Palestine, 2003
M.S.E., Kansas State University, 2008

AN ABSTRACT OF A DISSERTATION

submitted in partial fulfillment of the
requirements for the degree

DOCTOR OF PHILOSOPHY

Department of Computing and Information Sciences
College of Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas
2014

Abstract

Intrusion analysis, *i.e.*, the process of combing through Intrusion Detection System (IDS) alerts and audit logs to identify true successful and attempted attacks, remains a difficult problem in practical network security defense. The primary cause of this problem is the high false positive rate in IDS system sensors used to detect malicious activity. This high false positive rate is attributed to an inability to differentiate nearly certain attacks from those that are merely possible. This inefficacy has created high uncertainty in intrusion analysis and consequently causing an overwhelming amount of work for security analysts. As a solution, practitioners typically resort to a specific IDS-rules set that precisely captures specific attacks. However, this results in failure to discern other forms of the targeted attack because an attack's polymorphism reflects human intelligence. Alternatively, the addition of generic rules so that an activity with remote indication of an attack will trigger an alert, requires the security analyst to discern true alerts from a multitude of false alerts, thus perpetuating the original problem. The perpetuity of this trade-off issue is a dilemma that has puzzled the cyber-security community for years.

A solution to this dilemma includes reducing uncertainty in intrusion analysis by making IDS-nearly-certain alerts prominently discernible. Therefore, I propose *alerts prioritization*, which can be attained by integrating multiple methods. I use IDS alerts correlation by building attack scenarios in a ground-up manner. In addition, I use Dempster-Shafer Theory (DST), a non-traditional theory to quantify uncertainty, and I propose a new method for fusing non-independent alerts in an attack scenario. Finally, I propose usage of semi-supervised learning to capture an organization's contextual knowledge, consequently improving prioritization. Evaluation of these approaches was conducted using multiple datasets. Evaluation results strongly indicate that the ranking provided by the approaches gives good prioritization of IDS alerts based on their likelihood of indicating true attacks.

HANDLING UNCERTAINTY IN INTRUSION ANALYSIS

by

LOAI M. M. ZOMLOT

B.S., Islamic University, Palestine, 2003
M.S.E., Kansas State University, 2008

A DISSERTATION

submitted in partial fulfillment of the
requirements for the degree

DOCTOR OF PHILOSOPHY

Department of Computing and Information Sciences
College of Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas
2014

Approved by:

Major Professor
Xinming Ou

Abstract

Intrusion analysis, *i.e.*, the process of combing through Intrusion Detection System (IDS) alerts and audit logs to identify true successful and attempted attacks, remains a difficult problem in practical network security defense. The primary cause of this problem is the high false positive rate in IDS system sensors used to detect malicious activity. This high false positive rate is attributed to an inability to differentiate nearly certain attacks from those that are merely possible. This inefficacy has created high uncertainty in intrusion analysis and consequently causing an overwhelming amount of work for security analysts. As a solution, practitioners typically resort to a specific IDS-rules set that precisely captures specific attacks. However, this results in failure to discern other forms of the targeted attack because an attack's polymorphism reflects human intelligence. Alternatively, the addition of generic rules so that an activity with remote indication of an attack will trigger an alert, requires the security analyst to discern true alerts from a multitude of false alerts, thus perpetuating the original problem. The perpetuity of this trade-off issue is a dilemma that has puzzled the cyber-security community for years.

A solution to this dilemma includes reducing uncertainty in intrusion analysis by making IDS-nearly-certain alerts prominently discernible. Therefore, I propose *alerts prioritization*, which can be attained by integrating multiple methods. I use IDS alerts correlation by building attack scenarios in a ground-up manner. In addition, I use Dempster-Shafer Theory (DST), a non-traditional theory to quantify uncertainty, and I propose a new method for fusing non-independent alerts in an attack scenario. Finally, I propose usage of semi-supervised learning to capture an organization's contextual knowledge, consequently improving prioritization. Evaluation of these approaches was conducted using multiple datasets. Evaluation results strongly indicate that the ranking provided by the approaches gives good prioritization of IDS alerts based on their likelihood of indicating true attacks.

Table of Contents

List of Figures

List of Tables

Acknowledgments

I would like to express my gratitude to my advisor, Dr.Ou, for his continual encouragement and assistance over these years. I have learned so much from his work and I am grateful to have had the opportunity to work closely with him. I look forward to continue collaboration with him.

I would like to thank my committee members for their participation in this process and for their comments and suggestions. I am particularly grateful to Dr.Rajagopalan, whose insight and knowledge have been invaluable.

At last, I would like to thank everyone who helped me throughout this journey - the faculty, staff, and students. I am grateful to each professor who has instructed me over the past years. For my friends in Argus group, I am grateful for the time we have spent in cooperative achievements.

This research was supported by the U.S. National Science Foundation (NSF) under award number 0954138 and 1018703, Air Force Office of Scientific Research (AFOSR) under award number FA9550-09-1-0138, and HP Labs Innovation Research Program.

Dedication

To my wife, parents, and family.

Chapter 1

Introduction

Intrusion-Detection System (IDS) has been seen as the "silver bullet" that ensures security of an enterprise network against conceivable attacks. Despite the wide spread of this technology, it is not effectively used because of the large amount of false alarms that it produces. For example, the well-known open source IDS system Snort[89] is run on a production network with just a couple of hundred machines and it produces hundreds of thousands of alerts on a daily basis, a majority of which are false alarms. In big enterprises the problem worsen. IDS deployment often creates massive amount of events that typically flow into the enterprise's Security Operation Center (SOC), consequently causing an unrealistically overwhelming amount of work and long working shifts for security analysts. Throughout my experience of working with and interviewing many security analysts, I discovered that an event discerned to be potential attack or false alarm, in 10 minutes interval. In many teams, this number decreases to one minute or less. This unrealistic short time forces the team to sample from the IDS events list. The short-term solution may be to recruit additional staff, but this approach cannot help even in the near future, especially because of the rapid growth of network size, which puts the IDS effectiveness on the line.

To technically cope with this problem, practitioners typically create specific IDS rules (signatures) that precisely capture very specific attacks and reduce the overall false-positive rate. However, this results in failure to discern other attacks or other forms of the targeted attack because of the polymorphic nature of the attacks, which is a result of the human intelligence that stands behind them. On the other hand, to prevent false negatives, *i.e.*, detection misses, the IDS' rules engineers resort to combine the above approach with a more generic one, so that an activity with even a remote possibility of indicating an attack will trigger an alert. It then becomes the responsibility of a security analyst monitoring the IDS output to distinguish the true alarms from the large number of false ones, thus perpetuating the original problem. For example, Internet Control Message Protocol (ICMP) packets are routinely generated to indicate networking errors (*e.g.*, non-existence of a service). This could be an indication that a remote attacker is probing a host or simply be a benign error. Many IDSes capture these "ICMP" packets as potential malicious activities, thus unavoidably triggering a large number of false positives. Therefore, an inevitable trade-off between false positive and false negative becomes a dilemma puzzling cyber-security community.

The origin of this problem is deeply rooted into the base-rate fallacy (paradox) phenomenon, which affects any detector. This phenomenon reveals itself when one probabilistically tries to detect an attack in a big volume of benign traffic, consequently leading to the declaration of many of the benign traffic as the culprit. However, to avoid this problem, detection accuracy must exceed the rarity of the attack, which is not feasible technically. The prevalence of this problem in intrusion analysis was first noted by Axelsson[18] and since then, accurate detection of intrusion by a single sensor has become virtually impossible. A very low false positive rate results in so many false alarms as to make the analysis useless in practice. This problem made the IDS acts like the boy who cried wolf in the famous story, but even in a worse way because of the large amount of alerts in the era of big data. This created a behavior various the practitioners call *cognitive bias*, *i.e.*, the tendency to ignore most IDS events (or sometimes all). As a result, it is common among practitioners to altogether disable IDS signatures that tend to trigger large amount of false positive. In fact, security analysts often tend to not use the standard IDS rule sets, but instead resort to secret (unpublished) attack signatures that are highly specific to their experience and environment. Consequently, security analysts risk using these secret precise signatures sets because they help capture only "low-hanging fruit", and many attacks are likely missed due to disabled signatures. Turning off IDS signatures is like turning a blind eye to attack possibilities. Unfortunately, practitioners see no alternative, with the lacking of any other significant distinguishing feature between the alerts.

1.1 Thesis

All above have instigated the problem of intrusion analysis, or the process of examining real-time events such as IDS alerts and audit logs to identify and confirm successful attacks and attack attempts into computer systems. In this dissertation, I aim to investigate and propose a technical approach to help automate the intrusion analysis process. This work considers human effort in relation to the problem, thereby reducing the burden from the security analyst. Therefore, I propose *events prioritization*, achieved using reason under uncertainty and machine learning approaches, to reduce a security analyst's workload.

Thesis. *The prioritization of IDS-correlated-alerts scenarios using a customized version of Dempster-Shafer theory and machine learning can reduce uncertainty in intrusion analysis.*

Prioritization is achieved using multiple approaches that handle uncertainty in intrusion analysis:

- Alert correlation to establish the attack scenario from small pieces of evidence in a ground-up manner
- Customization and application of a mathematical theory, *i.e.* Dempster-Shafer Theory (DST) in this work, to quantify uncertainty in attack scenarios
- A machine learning approach to automatically capture the contextual knowledge specific to an enterprise's network

This dissertation primarily focuses on these three approaches to handle uncertainty in intrusion analysis. Chapter 2 explains my approach in correlating IDS alerts and Chapter 3 elaborates on my approach using DST to quantify uncertainty in alert correlations. Chapter 4 demonstrates usage of machine learning in this problem. Finally, Chapter 5 contains the conclusion.

1.2 Intrusion Detection Systems

Cyber-attacks on homes, businesses, and governments have become daily events which violate, Confidentiality, Integrity, and/or Availability (CIA) of the affected computer systems. Therefore, a system must be in place that can detect and/or prevent these attacks on a computer host or network. Consequently, many methods and systems have emerged to automate this process.

Definitions[19,57]

- *Intrusion*: attempt to compromise Confidentiality, Integrity, and/or Availability (CIA) in a computer system or network
- *Intrusion Detection*: process of monitoring events occurring in a computer system or network and analyzing them for signs of intrusions
- *Intrusion Detection System (IDS)*: piece of a software or a hardware system that automates the intrusion detection process
- *Intrusion Prevention System (IPS)*: system containing all IDS capabilities but can also actively stop possible incidents.

Currently, IDS is widely used organizational networks. IDS can be categorized into network-based and host-based and categorized by approach into signature-, anomaly-, and specification- based. These categorizations are detailed in the following

IDS by Deployment

- *Host-Based IDS (HIDS)*: system that resides as an agent on the local computer and monitors machine behavior, *e.g.*, by examining the logs

- *Network-Based IDS (NIDS)*: system that monitors network traffic generally consisting of sensors distributed over the network and a processing unit. The sensors sniff network packets, *e.g.*, TCP/IP packets, and the system attempts to identify malicious packets or anomalous activity on the network.[52]

Detection Approaches

Intrusion detection approaches typically fall into one of the following categories:

- *Signature-based or Misuse-based IDS*

  A signature is a pattern that corresponds to a known attack or threat; misuse detection is the process to compare patterns against captured events to recognize possible intrusions. For example, in network IDS the packet content has a known pattern of attack, such as NOP sled or shell code[52]. The primary advantage of this approach is the ability to precisely detect known attacks, but it is unable to detect previously unseen attacks. Therefore, evasion of these signatures is inventible, requiring users of this type of IDS to write generic signatures in order to minimize the attackers' evasion effect, resulting in additional false positive.

- *Anomaly-based IDS*

  An anomaly is a deviation from a known or expected behavior on the computer system or network. Normal behaviors are derived from monitoring regular activities for the network and hosts over a period of time and establishing a normal profile. Anomaly-based IDS compares normal profiles with observed events to recognize any outlier as an attack. For example, if a user's work hours are always between (8:00 am - 5:00 pm), and he/she suddenly diverges from these times the IDS will raise an alert. Multiple failed login attempts are also considered anomalous behavior. This approach, unfortunately, cannot detect slow, gradual behavior divergence, but the primary advantage of this approach is its ability to detect novel attacks. However, it suffers from high false alarms because previously unseen legitimate system behaviors are also recognized as anomalies and flagged as potential intrusions. Due to the diversity and dynamism of the networks and applications, establishing a normal baseline is also difficult, consequently hindering usage of anomaly-based IDS in practice.

- *Specification-based IDS*

  In this approach, manually developed specifications are used to characterize legitimate program behaviors. This approach depends on vendor-developed generic profiles to specific protocols that enable it to trace protocol states. In general, the network protocol models in a specification-base IDS are based on protocol standards from international standard organizations, *e.g.*, Internet Engineering Task Force (IETF). The advantage of this approach is that it does not generate false alarms when legitimate unusual user behaviors are encountered. It can also detect previously unknown attacks because of its ability to detect attacks that deviate from the specified-legitimate behaviors. However, specification development for such a system needs significant effort, which affects approach usability. In addition, the effectiveness in reducing false positive is still questionable.

- *Hybrid IDS*

  The hybrid approach is achieved with the use of multiple methodologies to provide the best extensive and accurate detection. For example, anomaly-based and signature-based are complementary methods by which IDS can cover unknown and known attacks/threats.

1.3 Intrusion Analysis

Intrusion analysis answers the questions, *Are all these alerts important?* or *Is this alert true or false positive?* Therefore, intrusion analysis can be defined as follows:

- Organizing and characterizing data regarding user and system activity in order to identify activity of interest[20].
- Process of identifying attack traces from large amounts of system monitoring data, *e.g.*, IDS alerts and audit logs, either on the fly or offline[67].

Intrusion analysis is a combined approach that utilizes intrusion detection and computer forensics to achieve its goal.

Approaches in Intrusion Analysis

Currently, Intrusion analysis can be divided into two approaches: the primitive approach and alert correlating approach. In the primitive approach, the security analyst manually and randomly goes through raw IDS alerts and logs with hope to identify real attacks. This approach may work if the security analyst monitors a small networks with tens of machines. In the alert correlating approach, the security analyst attempts to reach conclusions through events correlation. Correlation can be defined from statistical perspective as a technique that indicates whether and how strongly pairs of variables are related, *e.g.*, correlation between the demand for a product and its price. Thus, correlation can be used to find relationships between multiple IDS alerts in order to build attack scenarios in ground-up manner. For example, by correlating a port scan action from an external machine toward an internal server and malicious behavior of the latter, the security analyst can infer that a hidden connection is present between these two events, which means the external machine has exploited the server.

Therefore, IDS alert correlation can be defined as the reconstruction of high-level incidents from low-level events. This approach has been used to remediate the high false-positive problem in practice. Investigation of at multiple observation points and correlating events can potentially reduce the false-positive rate and increase the confidence in an attack scenario. Axelsson's[18] reasoning implies that useful intrusion detection is impossible to achieve based on a single event, such as a network packet, because features within the event are too limited to provide necessary differentiating power to extract an extremely weak attack signal from background traffic. Therefore, the reasonable assumption can be made that in order to make IDS sensors useful, events from multiple sensors must be correlated to increase features available for decision making.

1.4 Uncertainty

Uncertainty is the quality or state of being uncertain, or something that is doubtful or unknown[15]. Two types of uncertainty are prevalent.

- Aleatory or Objective Uncertainty

  This uncertainty is due to variability of input or system parameters when characterization of variability is available. It results in from random system behavior, *e.g.*, sensor errors. This type can be handled by the frequentist approach associated with traditional probability theory[47], which is the limit of an event's relative frequency in a large number of trials.

- Subjective or Epistemic Uncertainty.

  This uncertainty is due to variability of input or model parameters when corresponding variability characterization is not available, or it is uncertainty due to the lack of specific knowledge in system analysis. For example, the interpretation of a potentially malicious event that also has a benign explanation is a common example of this type of uncertainty. The application of traditional probability theory here runs into fundamental problems because probabilistic analysis requires an analyst to have information on prior probability of all events, which is often not available. Therefore, the uniform distribution function, is often used to represent the unknown, which is justified by the Principle of Insufficient Reason or Indifference[78]. Consequently, all simple events for which a probability distribution is not known in a given sample space are equally likely. While application of such a principle may be appropriate for physical systems, it is questionable in an area such as intrusion detection in which beliefs and human-built models are innate to the analysis. This is especially when a frequentist approach to estimate probabilities is infeasible or available information can be ambiguous or conflicting. For example, difficulties arise when an analyst attempt to assign a probability to the event that a particular nuclear power plant will experience a meltdown. What are the set of possible events, and is it correct for these elements to be equally likely? An approach to this problem is well established in the various non-traditional theories for uncertainty, such as Dempster-Shafer Theory, Subjective Logic, and Possibility theory[47].

1.5 Related Work

Events Correlation

Intrusion analysis or IDS alert correlation has been extensively studied in the past ten years[29,30,34,35,62,64–67,87,99,100,103,106,109]. *Cheung, et al.*[30] propose the construction of a set of modules that describe specific attacks with pre-conditions that must satisfied for the attack to occur, the attack activity itself, and the post-condition that may result if the attack succeeds. To link two modules, the post-condition of one module must match the pre-module of another. To model these multi-stage attacks, they developed a model called Correlated Attack Modeling Language (CAML) that aims to develop attack patterns. They also used attack patterns developed by them and stored in a knowledge base. *Ning, et al.*[64] proposed a similar approach using pre- and post-condition of attacks using predicates. Their implementation maps alerts to an internal representation called hyper-alert. After mapping each raw alert to one of the pre-generated hyper-alerts stored inside a knowledge base, an attack scenarios are created by correlating hyper-alerts. Final visualization of the attack scenario is presented as a graph using the GraphViz tool. Concerns regarding these two works are the adoption a pre- and post-condition correlation module. A potential disadvantage of ascribing a pre- and post-condition to an IDS alert is that the model utilizes specific attack patterns, thus risking the potential of high false negatives. In addition, maintaining the knowledge base for new attack types is an overhead, and no evidence indicates how well their approach will work on production systems in which a large volume of false positive alerts is typically received.

Ren, et al.[72] proposed the usage of a real-time correlation system. They use an online component that receives and groups alerts into hyper-alerts while utilizing a knowledge base to constructs an attack graph. A hyper-alert is representative on a group of similar alerts, and the knowledge-base is maintained by an offline component. This component maintains tables that specify the frequency of occurrence of possible hyper-alerts by types and the correlation between different hyper-alert pairs. The component is dynamically updated depending on network traffic observed over a past time-window. In their work, hyper-alerts were attack-specific, and updating the knowledge-base would be tedious considering the rate at which new exploits are developed. In addition, automatic knowledge base construction is limited to past attacks patterns and is not able to detect new ones.

The primary work from which I built my research and implementation is by *Ou, et al.*[67]. In this work, the authors proposed an empirical approach to model uncertainty using an IDS alert correlation tool called SnIPS[54]. SnIPS builds dynamic abstract proof traces supported by evidences. To build a proof trace, they developed a generic knowledge-base derived from attacker's intentions and not on specific attacks. Attacks reflect attackers' intentions and so by carefully modeling attacker behavior through a set of generic rules, low-level IDS alerts can be appropriately mapped to their high-level counterparts, resulting in a simple and concise set of rules that is easy to maintain. SnIPS has the ability to elevate confidence in proof traces by automatically joining two different proofs and strengthening them using their certainty tags. However, I found that proof strengthening is not sufficient, because the strengthening rules seems to be ad-hoc. Therefore, in my work I built a correlation engine that auto-correlates and combines SnIPS proof traces and produces a complete attack scenario bounded by a given time-window. This scenario is visualized as an inference graph that is easy to navigate by the security analyst. Finally, in their work they assume that the strengthened two proofs are independent, meaning they are based on disjoint sets of observations, which is a strong assumption to make, especially in a big attack scenarios.

Dempster-Shafer Theory

DST is one of multiple theories that handle the second type of uncertainty, *e.g.*, Dempster-Shafer theory, Subjective Logic, Fuzzy Logic and Possibility theory, some of which have been proposed in IDS alert fusion[92]. According to *Sentz*[80] DST is superior to other theories in this application because of its relatively high degree of theoretical development. DST is a generalization of traditional probability theory in which probabilities are assigned to sets of events as opposed to mutually exclusive singletons events. In addition, it provides a tool to combine different types of evidence from multiple sources. Finally, it is widely applied in sensor fusion.

Chen, et al.[28] proposed a general approach of applying standard DST to combine multiple sensor reports for intrusion detection in ad-hoc networks. *Yu, et al.*[103,104] extended DST to handle alert fusion in IDS alert correlation systems. They observed that direct application of DST in IDS alert fusion provides non-intuitive results. *Sun, et al.*[90] proposed the application of DST to risk analysis of information systems security. They presented an evidential reasoning approach that provides a structured model to incorporate relevant risk factors, related countermeasures, and their interrelationships when estimating information system risk. *Tang, et al.*[95] applied DST to fault diagnosis in overlay networks. However, neither of them addressed the crucial issue of non-independence among evidence sources, which I discuss in Chapter 3.

Denceux[38] proposed an approach for handling the combination of non-independent sources in DST. He pointed out that lack of independence in evidence is a valid concern in many applications; so he proposed a new rule of combination, the "cautious rule", to handle this issue. The cautious rule is designed to be as general as possible and is hence very complex. Conversely, customization of the combination rule, in my work, follows the general idea proposed by Shafer[85] and is based on a simple probabilistic semantics. Therefore, it could be considered a highly specialized case of the general cautious rule appropriate to this problem.

There have also been work on using Bayesian Networks (BN) in intrusion detection and IDS alert correlation[16,61,107]. To use BN approach, the user must have prior probabilities of events, that are often unavailable. DST does not have this requirement and DST can quantify the *unknown*. Therefore, DST can be seen as a way to generalize probability theory. *Cole*[32] studied the problem of multi-step attack detection in the presence of uncertainty in IDS parameters and pointed out to the importance of considering uncertainty when designing IDS. *Chen, et al.*[27] proposed an application of DST to the detection of anomalies in a variety of systems, such as worm detection in email and learning in biological data. They showed that a combination of multiple *independent* signal sources allows the possibility to achieve better results than by using a single signal. They pointed out that the advantage of using DST over Bayesian is that no *a priori* knowledge is required, making it potentially suitable for anomaly detection of previously unseen information. Bayesian inference requires *a priori* knowledge and does not allow allocation of probability to ignorance.

Guofei, et al.[45] proposed an alert fusion technique based on likelihood ratio test (LRT). However, in their model, prior probability of an attack should always be predefined. *Barreno, et al.*[22] introduced an optimal approach for combining binary classifiers using Neyman-Pearson lemma. However, it is not clear if these approaches are effective in the IDS problem.

Machine Learning

Pietraszek[68] used machine learning to classify IDS alerts into true and false positives. In a later work, *Pietraszek and Tanner*[69] proposed a more complete system in which noisy alerts are eliminated before feeding them into the system. This line of work differs from mine in two ways: it operates at the IDS alerts level and utilizes RIPPER rule learner. The rule learner approach has an explicit classification logic which allows a human expert to inspect the classifier and verify its correctness. Instead of a rule learner, I used semi-supervised learning with Support Vector Machine (SVM) algorithm to classify interesting and non-interesting correlation graphs. In my framework, the classifier adapts according to new incoming labels and presents non-intuitive conflicting results to the expert for further study. Moreover, the RIPPER rule learner is typically applied before the correlation stage, but my approach uses correlation graph-related features to build the classifier. Finally, in my approach noisy alerts are not removed because the noisiest alerts could potentially have a link to a true attack.

Beaver, et al.[23,93] proposed an approach using "in-situ" learning and semi-supervised learning to build a model for intrusion detection. Attack traffic is introduced to the network where the detection tool is deployed, and this labeled data is used to train the classifier. Their model, as opposed to anomaly detection, learns known attacks and normal traffic. It remains to be seen how effective the approach when the system is deployed in production networks. My application of machine learning had a different objective. Instead of using machine learning to make a decision as to whether or not an event is malicious, I used it to prioritize alert-correlation graphs from an up-stream analysis tool, in an attempt to save security analysts' time. I also conducted my evaluation in a live production network.

Bolzoni, et al.[25] proposed a system that automatically classifies attacks, *e.g.*, buffer overflow or SQL Injection, detected by an anomaly-based network intrusion detection system. This is done by comparing extracted byte sequences from an alert's payload to previously collected data, *e.g.*, Snort alert classification. The goal of my approach is to classify alert-correlation graphs into "interesting" and "non-interesting", where "interesting" means that security analyst will need to conduct further analysis.

Chiu[31] used semi-supervised learning to reduce false alerts from IDS. They introduced a method using TCP information of network connections to reduce false alarms. They use semi-supervised learning technique called Two-Teachers-One-Student (2T1S) to gain more useful information from the large amount of unlabeled data. The system was only tested on DARPA 1999, the usage of which has been severely criticized[60] in machine learning applications. It remains to be seen how effective the approach will be when deployed in production networks.

In addition to the mentioned limitations in the previous works, my application of machine learning is on top of intrusion analysis. The output of intrusion analysis is a list of automatically constructed correlation graphs with confidence metric. This metric is a measure of DST belief that the system has in graph depending on provided evidences.

There has been a long line of work on the application of machine learning in anomaly-based intrusion detection[39,43,48,49,55,74–76,86,108]. It has been pointed out that significant challenges exist in applying machine learning in this area[88]. However, my application of machine learning had a different goal than past works. My machine-learned model helps the security analyst prioritize alerts correlations from an intrusion analysis system, which relies on (multiple) IDS systems. My method is *not* to build an intrusion detector through machine learning. Therefore, the application of machine learning is justified due to the nature of the problem.

1.6 Contributions

As mentioned, prioritization of IDS-correlated-alerts scenarios using a customized version of DST and machine learning can reduce uncertainty in intrusion analysis. Therefore, my contributions, listed below, focus on this thesis.

1. Design and implement an alert correlation engine[91](Chapter 2).

    - The engine generates attack scenarios or alert correlations within a given time window.

- The correlation engine can report attacks in real time by continuously monitoring network traffic.
- Present the correlation-engine's output in multiple formats using a web-interface.

2. Applying DST to prioritize attack scenarios[111](Chapter 3).

- Use "unknown" to capture sensor quality.

  DST allows specification of a weight for "unknown" rather than specifying precise probabilities for every possible event in the space by using the uniform distribution function. This allows to represent lack of knowledge or ignorance to capture the intuitive notion of IDS sensor quality.

- Account for lack of independence among IDS correlated alerts.

  DST has the assumption of independence in evidence sources, a property hard to justify in practice, especially in IDS alert fusion problem since many alerts are triggered by identical or similar signatures. In this problem, in order to derive the overall belief on attack status, such non-independence must be appropriately accounted for. I developed a customization for Dempster's rule of combination, specialized to my alert fusion problem. To the best of my knowledge, this is the first sound DST non-independent rule in *the IDS alerts fusion* application.

- Efficient algorithm.

  A direct application of DST formulas results in exponential (in the number of IP addresses in the graph) blow-up of belief combinations. I adopted a "*translate-then-combine*" approach so that beliefs are propagated in a correlation graph and combined only at join points in the graph, thus producing an efficient algorithm with worst-case running time quadratic in the number of IP addresses in alerts.

- Robustness of solution.

  The goal of work is to prioritize alerts by confidence, so the relative order of hypotheses is the matter here, as opposed to establishing absolute certainties regarding attacks scenarios. My application of DST requires the assignment of numeric values to IDS sensors to act as weights for their alerts. However, there is no help in the theory itself as to the manner of assignment. The weights of evidences may affect the final conclusions in standard DST. Since I am interested only in *relative* belief strengths assigned to hypotheses, this approach is robust to small changes in these weights as long as the final ranking is not significantly impacted. That is, for any two ranked hypotheses, absolute belief values are irrelevant as long as the relative strengths of belief remain unchanged when slightly varying the numeric parameters. Experimental analysis shows that this is true; the classifier's operating characteristic does not change when the weights' values are varied within a small range.

3. Apply machine learning as a complementary approach to aid in automating the process of intrusion analysis[110] (Chapter 4).

   - My method minimizes the time and effort of training the model in the deployment stage by using the security analyst's effort to investigate the correlations' validity to produce labeled data. In addition, I demonstrate that the usage of semi-supervised learning enables the model to be in use with as low as 10% of the required dataset size for supervised learning.
   - Creating my own dataset by collecting real production network traffic and labeling it, and share lessons learned from applying machine learning in intrusion analysis problem.

Chapter 2

Using Events Correlation to Reduce Uncertainty in Intrusion Analysis

Intrusion events correlation, *i.e.* the reconstruction of high-level incidents from low-level events, has been used to reduce uncertainty in intrusion analysis. In practice, security analysts typically use multiple observation points and correlate their events so they can potentially reduce the false-positive rate and increase confidence in intrusion analysis. Axelsson's[18] reasoning implies that the achievement of useful intrusion detection based on a single event, such as a network packet is difficult, since features existing in the event are too limited to provide differentiating power to extract extremely weak attack signal from background traffic. Therefore, the reasonable assumption can be made that in order to make IDS sensors useful, events from multiple sensors must be correlated to increase the available features for decision making.

In the following sections, I elaborate on my approach of correlating events in intrusion analysis. I use SnIPS[67] correlation approach as a foundation for my correlation engine implementation. SnIPS' output is a list of proof traces, and each trace supports a hypothesis. These proof traces are used as input to my correlation engine. Output of the correlation engine is a list of attack scenarios supported by evidences, *e.g.* IDS events. The attack scenario is a logical inference graph with one or more sink node(s). The sink node is the frame of interest or the hypothesis that shows a specific machine is compromised; all inference paths are proofs traces that support this hypothesis.

The following section provides a background of SnIPS and an elaboration of my approach in correlating events to generate a correlation scenario. Finally, I elaborate on the implementation and the testing.

2.1 Snort Intrusion Analysis using Proof Strengthening

I decided to use the open-source framework SnIPS[67] as a foundation to build and test my thesis. SnIPS was developed and maintained in Argus Lab in the department of Computing and Information Sciences (CIS) at Kansas State University (KSU), Manhattan, Kansas. SnIPS works on top of IDS sensors and audit logs to further analyze reported events in order to identify possible incident scenarios. Figure 2.1 illustrates SnIPS. The intuition behind SnIPS is to mimic the thinking process of security analyst during incidents investigation. It maps raw observations, *e.g.*, IDS alerts and syslog, to their semantics and then it reasons about them using a succinct internal inference rules to reach a final conclusion.

Currently, SnIPS works with the widely used open source Snort IDS system[89] network intrusion prevention and detection system (IDS/IPS). SnIPS compares a network packet with a set of predefined signatures (Snort rules) that specify certain patterns often associated with malicious activities. Furthermore, SnIPS maps the trustworthiness of each Snort rule to a discrete tag, *e.g.*,"possible","likely",or "certain".

Ou *et al.*[67] demonstrated that building the mappings can be done with minimal overhead since the rules-related information already exists in an ad-hoc manner in IDS-signatures' documentation, such as Snort rule repository, which can be automatically analyzed to extract the mappings. These tags were used in my research to derive a Snort rule's quality metric. To generate an attack trace, IDS events must pass through multiple stages within SnIPS. I discuss these stages, using their order in SnIPS, in the following sections.

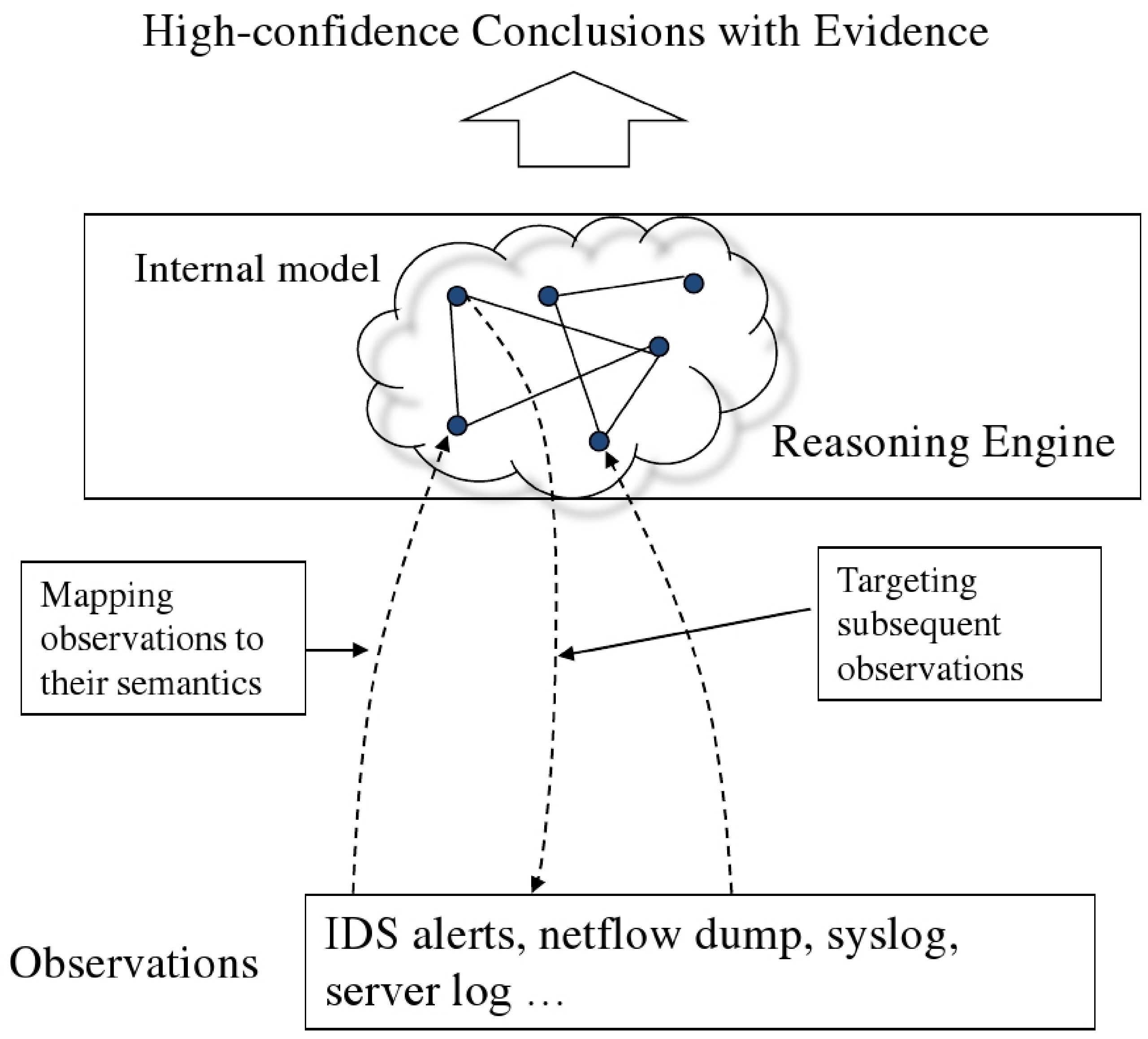

High-confidence Conclusions with Evidence
Internal model
Reasoning Engine
Mapping observations to their semantics
Targeting subsequent observations
Observations
IDS alerts, netflow dump, syslog, server log …

Figure 2.1: *Snort Intrusion Analysis using Proof Strengthening (SnIPS)*

2.1.1 Semantic Pre-processing

SnIPS' pre-processing stage is performed to translate and reduce the amount of information entering the reasoning engine. This process consists of the following parts:

Observations Mapping

SnIPS reads Snort's alerts as main input and maps the observations (events) to their semantics (meaning). This process utilize a set of mapping rules called *obsMap*.

Definition 1. Observations $\xrightarrow{mode}$ Internal Conditions

Definition 1 gives the formal mapping rule between observations such as IDS alerts, and abstract meaning (semantics). The *mode* is used as a tag indicating strength of the belief (*e.g.*, *possible, likely, or certain*). The assignment of the mode is created by interpreting the natural language description of Snort rule (signature). Example 2.2 shows an obsMap rule that maps a *port scan* alert to *probing* activity with the *possible* mode. In addition, the example shows additional mapping between *shell code* detection in the traffic and *send exploit*. This mapping is *certain* mode extracted from Snort-rule's description. Therefore, these rules are correspondence mappings between what the *security analyst sees* and *what he/she knows*.

Summarization

SnIPS' summarization stage is performed to reduce the amount of information entering the reasoning process. SnIPS applies data abstraction technique by grouping a set of similar predicates into a single "summarized" one. The summarization is done on timestamps and IP addresses. To summarize timestamps, if a set of internal conditions differs only by the timestamp, SnIPS merges them into a single summarized internal condition with a time range between the earliest and latest timestamp in the set. For further reduction, SnIPS is able to summarize the external IP address into an "external" variable, as needed. SnIPS maintains mapping between summarized "predicates" and raw observations in a backend

$$\begin{array}{l}
\textit{obs(portScan(H1,H2))} \xrightarrow{possible} \textit{int(probeOtherMachine(H1,H2))} \\[1em]
\textit{obs(shellcodeDetected(H1,H2))} \xrightarrow{certain} \textit{int(sendExploit(H1,H2))} \\[1em]
\textit{obs(memoryDumpMaliciousCode(H)} \xrightarrow{likely} \textit{int(compromised(H))} \\[1em]
\textit{obs(memoryDumpIRCSocket(H))} \xrightarrow{likely} \textit{int(compromised(H))} \\[1em]
\textit{obs(netflowBlackListFilter(H, BlackListedIP))} \xrightarrow{likely} \textit{int(compromised(H))}
\end{array}$$

Figure 2.2: *Example of mapping observations to their semantics: "obs" means observation; "int" means internal condition predicate*

$$\left.\begin{array}{c}
int(probeOtherMachine(ext_1, H), m, T_1) \\
int(probeOtherMachine(ext_2, H), m, T_2) \\
\vdots \\
int(probeOtherMachine(ext_n, H), m, T_n)
\end{array}\right\} int(probeOtherMachine(ext, H), m, range(T_1, T_n))$$

Figure 2.3: *Summarizing multiple predicates into one aggregated predicate: "int" means internal condition predicate: "ext_i" and "ext" mean external machines; "H" means a host machine; "m" means the mode of the rule; "T_i" means the timestamp for the internal condition*

database, thus helping to identify mapping of low-level observations and the summarized predicates in the next stages of SnIPS. Figure 2.3 illustrates the summarization process.

Black List IP Processing in SnIPS

SnIPS reads a black list of malicious machines as another feed to be correlated with Snort events. A machine can be added to the blacklist if it is found to be involved in malicious activities (e.g., bot activities, ssh brute-force attempts, etc.). Such a list can be used to map a blacklisted IP to the predicate *compromised*, with the *mode* assigned by the IP's age in the list. The IP address has a higher mode than the default one, if it is recently added to the list. Over time, confidence decreases because the infection will be eradicated from the machine. The blacklist can also be used to create a Snort rule that can be triggered whenever a communication exists between a local host and the black-listed IP. This alert is mapped using observation mapping rules, and the mode for this mapping is *certain* because it is a malicious communication. The second method is advantageous because it can capture all communication with a black-listed IP even if the IP did not trigger an alert.

2.1.2 Reasoning Engine

The goal of the reasoning process is to find all possible semantic links among summarized facts using a succinct *Internal Model*. Definition 2 gives the formal format for reasoning rules in the internal model or internal rules. The rule derives one internal condition from another with two qualifiers: *direction of inference* and *mode*. The direction tag has two values: *backward* or *forward*.

Definition 2. Condition 1 $\xrightarrow{direction\ of\ inference, mode}$ Condition 2

Figure 2.4 illustrates one internal rule. In forward inference, if machine H_1 is compromised, then it may perform malicious probing for another machine H_2. Conversely, if a machine H_1 is performing malicious probing against another machine, the inference can be made that machine H_1 is compromised (using the backward inference). Hence, each internal rule can be used in the forward or the backward direction throughout the reasoning process. Figure 2.5 shows an example of a proof chain using observation mapping and the reasoning engine.

The output of reasoning stage is a collection of individual *proof traces* (Figure 2.5) stored in SnIPS' backend database for the next stage. Each step is associated with a fact, such as *compromised*(H_1), and a time range *(startTime,endTime)*, indicating when the fact becomes true. The direction of inference (forward or backward) is also indicated in the proof trace. The time range of the conclusion can be calculated based on the time range of the antecedent and direction of the inference.

$$int(compromised(H_1)) \xrightarrow{forward,possible} int(probeOtherMachine(H_1, H_2))$$

$$int(sendExploit(H_1)) \xrightarrow{forward,likely} int(compromised(H_1))$$

$$int(compromised(H_2)) \xrightarrow{backward,possible} int(sendExploit(H_1, H_2))$$

$$int(probeOtherMachine(H_1, H_2)) \xrightarrow{backward,certain} int(compromised(H_1))$$

Figure 2.4: *Example of rules from the internal model: "int" means internal condition predicate; "H_i" means a host machine*

$$\begin{array}{l} int(compromised(H_1), likely) \\ \qquad\quad \Uparrow \\ \qquad\quad int(probeOtherMachine(H_1, H_2), likely) \\ \qquad\qquad\quad \Uparrow \\ \qquad\qquad\quad obs(portScan(H_1, H_2)) \end{array}$$

Figure 2.5: *Example of a proof trace: "int" means internal condition predicate; "H_i" means a host machine*

Proof Strengthening

SnIPS has the ability to elevate confidence in an attack scenario by combining proof traces in ad-hoc manner. Figure 2.6 is an illustration of the intuition behind the proof strengthening process, and Figure 2.7 is an example of strengthening two proofs. The first proof is from the previous example in Figure 2.5, and the second proof proves that if machine H_1 has a memory dump of malicious code, it is *likely* compromised. The strengthening of $likely_1$ and $likely_2$ results in the compromise of *certainty* of H_1.

I found proof strengthening is a special case of event correlation in intrusion analysis 1.3 because it just correlates a pair of proof traces. This approach is considered add-hoc if the goal to build holistic attack scenarios that span multiple evidences. In my approach, I built a correlation algorithm, as an alternative to the "proof strengthening" stage, on top of SnIPS' logical reasoner, which is discussed later in this dissertation.

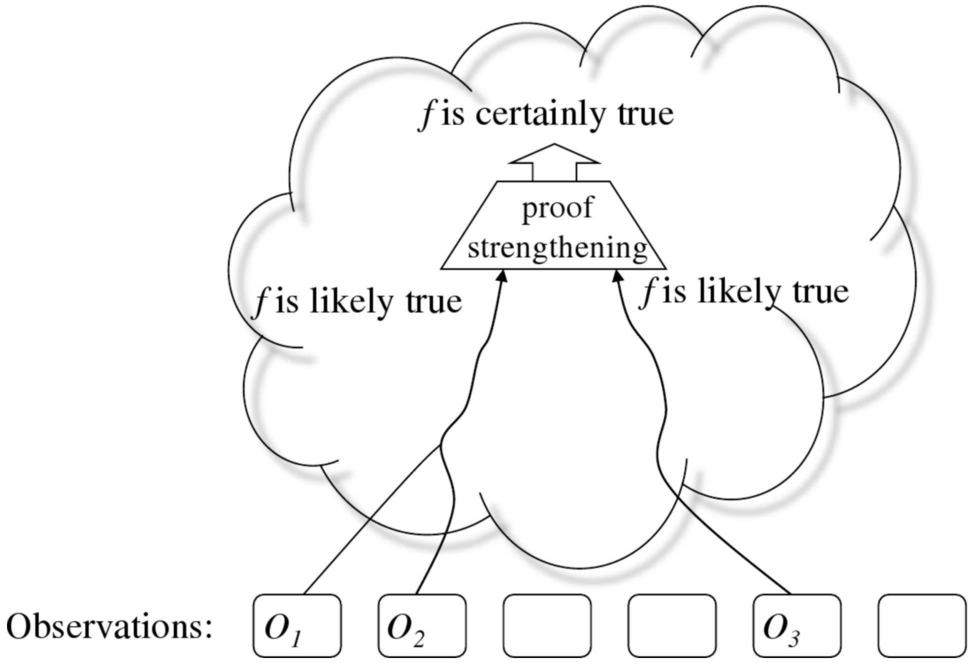


Figure 2.6: *Illustration of proof strengthening in SnIPS*

Strengthend hypothesis:
$int(compromised(H_1), certain)$

First Proof:
$int(compromised(H_1), likely)$
$\Uparrow$
$int(probeOtherMachine(H_1, H_2), likely)$
$\Uparrow$
$obs(portScan(H_1, H_2))$

Second Proof:
$int(compromised(H_1), likely)$
$\Uparrow$
$obs(memoryDumpMaliciousCode(H_1))$

Figure 2.7: *Proof strengthening in SnIPS: "int" means internal condition predicate; "H_i" means a host machine*

2.1.3 Dynamic Knowledge Base

The dynamic knowledge base is used in multiple stages previously described. It includes "obsMap Relations" used in the semantic pre-processing stage and "Internal Model" used in the logical reasoning stage. The dynamism of this model comes from the fact that it can be automatically or manually updated based on emerging threats. For example, black-listed IP addresses change every hour, and obsMap relations can be seamlessly updated accordingly.

2.2 Constructing Attack Scenarios

As mentioned, correlation intuitively reduce uncertainty in IDS output. The power of a correlation engine comes from the ability to reconstruct attack scenarios in a ground-up manner. Therefore, I chose SnIPS[67] as a foundation to build and test my approach in order to reduce uncertainty in intrusion analysis.

The output of SnIPS' logical reasoner is a list of *proof traces*. A proof trace partially covers an attack scenario, thereby serving as one source of information with which to reason. In cyber-security investigations, the security analyst must have a full scenario of an attack in order to comprehend the full picture of the attack. Therefore, I designed and developed a correlation engine that auto correlates and combines SnIPS' proof traces and produces a complete attack scenario bounded by a time-window. This scenario is visualized as an inference graph that can be navigated and queried. Instead of having a long list of raw and in-actionable alerts to process, the security analyst has a compact list of comprehensible correlations to validate every day.

Figure 2.8 shows an example of single-sink alert correlation graph automatically generated by the correlation engine. The correlation graph is a logical inference graph. SnIPS uses predicates, such as "*compromised*" , "*sendExploit*" , and "*probeOtherMachine*" , to describe various attack hypotheses. Five groups of alerts, $alert_1 - alert_5$, are triggered by four sensors. A sensor could be one IDS signature (*e.g.*, a Snort rule) or a group of IDS signatures that capture similar patterns. Sensor nodes (located in dotted squares) are not part of the graph but are added in Figure 2.8 for clarity. In this example, $alert_1$ is mapped to the fact that host ip_1 sent an exploit to ip_2; $alert_2$ and $alert_3$ are mapped to the fact that ip_2 did malicious probing to ip_3, and so on. The rationale for this correlation graph is that after ip_1 sent an exploit to ip_2 (Node 6), ip_2 could be compromised (Node 9). Once the attacker compromised ip_2, he/she could send malicious probing from there (Nodes 7 & 8). Therefore, these alerts are potentially correlated in the same underlying attack sequence. For representational simplicity, time information is not shown in the example but is included in the reasoning process, therefore, $alert_2 - alert_5$ occurred after $alert_1$. The arrows of the arcs indicate that all $alert_1 - alert_5$ support the hypothesis that ip_2 was compromised.

Section 2.2.1 elaborates the algorithm that computes the correlation graphs.

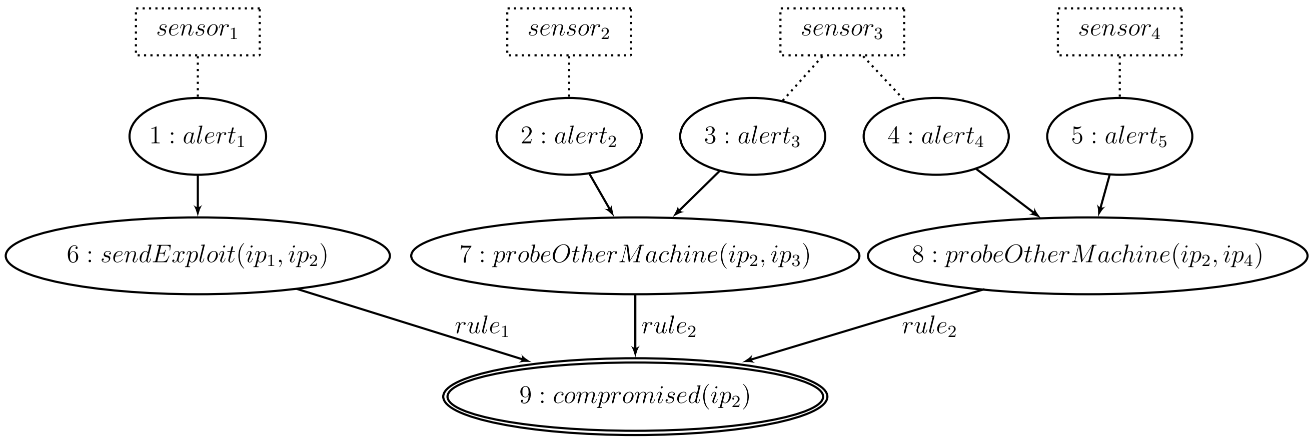


Figure 2.8: *Automatically generated correlation graph from the correlation engine*

2.2.1 Correlation Algorithm

I designed and implemented a correlation engine that auto correlates and combines SnIPS proof traces into a complete attack scenario bounded by a given time-window. For an input list of proof traces, the algorithm follows the following steps:

Step 1: Translate input proof traces into a form that can be handled by the engine, *i.e.*, object O_i, as shown in Figure 2.9. An object contains a number of fields, including the fact associated with it, the start time, and the end time. The object's fact and

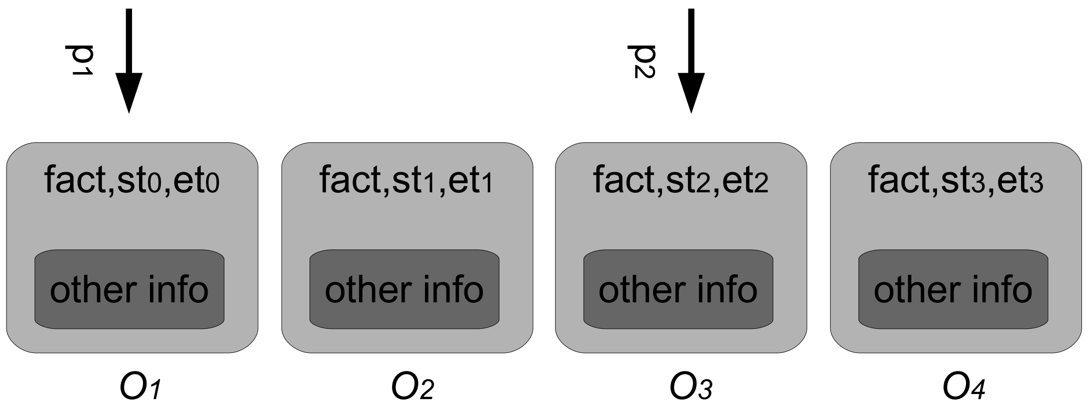


Figure 2.9: *Correlation of facts objects with overlapping time ranges*

times are derived from the final hypothesis in the trace, *e.g.*, *compromised*(H_1) in Figure 2.5.

Step 2: All objects (O_i) are classified based on associative facts. For example, all objects with the fact *compromised*(H) are in one group and so on. Figure 2.9 provides an example of one group. Let us assume that each fact in the figure has the form *compromised*(H). Each group of objects will be sorted ascendingly by the end time (et_i) and by the start time (st_j).

Step 3: Each group is correlated using overlapping time between objects. Figure 2.9 also illustrates the correlation process. Two sliding pointers track the correlation process. The first pointer p_1 starts at the first object O_1, and the second pointer p_2 moves to the second object. If the time range of O_2 overlaps with O_1, then the intersection of the two time ranges is calculated and stored in a variable $intTimeRange$. Pointer p_2 then moves to O_3 and calculates the intersection of its time range with $intTimeRange$ variable and updates the variable with the intersection time. The process stops when $intTimeRange$ becomes empty. The empty variable means that the object's facts can no longer be correlated. At this stage, a new graph node is created for all objects that have a non-empty time-range intersection. The created node has *fact,* $intTimeRange$ fields. After the node creation, p_1 moves forward until the time-range intersection for objects between p_1 and p_2 becomes non-empty. The graph *Nodes* are stored in a hash table; the key for this table is the object O of the *Node*. Therefore, the link of each graph's *Node* can be tracked.

Algorithm 1 shows the pseudo code for the correlation algorithm. Line 21 includes construction of the graph edges utilizing "other info" field in each object to connect merged nodes.

Algorithm 1: *Algorithm of building a correlation graphs*

Require: Parameter $P = ProofTraceList$, such that $P = \{p_0, p_1, \cdots, p_n\}$, where p_i is a proof trace.

```
1:  function CORR($P$)
2:      $ObjectsList \leftarrow$ parse all $p_i \in P$
3:      for each $Object(O)$ in $ObjectList$ do
4:          $ObjectsGroupList \leftarrow$ group by fact of Object $O$.
5:      end for
6:      for each $ObjectGroupList$ do
7:          sort ascendingly by the $endTime$,$startTime$ of the $TimeRange$.
8:          $Graph \leftarrow$ CREATEGRAPH($ObjectGroupList$)
9:      end for
10:     return $Graph$
11: end function
12:
13: function CREATEGRAPH($ObjectGroupList$)
14:     for each $Object(O_i)$ in $ObjectGroupList$ do
15:         $intTimeRange \leftarrow$ find the intersection of $timeRange$ with the next $O_i$
16:         if $intTimeRange$ is Empty then
17:             $NodesHash \leftarrow$ create new node with $intTimeRange$ and $O_i$'s fact and use $O_i$ as the key for the hash table.
18:         end if
19:     end for
20:     for each $Node$ in $NodeHash$ do
21:         $Graph \leftarrow$ build the edges from the related $O_i$
22:     end for
23:     return $Graph$
24: end function
```

2.3 Implementation and Evaluation

This section describes implementation and evaluation of the correlation algorithm.

2.3.1 Implementation

Semantic mapping and logical reasoning of SnIPS are implemented using the Prolog system XSB[70]. SnIPS engine output is a list of Prolog files, therefore, I introduced a back-end database to store all proof traces, which needed a change in the implementation of SnIPS, so I utilized MySQL, the open source database. The correlation algorithm is implemented in Java. The correlation engine reads input proof traces from MySQL and outputs a list of correlation graphs, which are stored in the back-end database.

Moreover, I used a web interface visualizer that I implemented in PHP and HTML. Graphs visualization was generated using the Graph Visualization Software (GraphViz)[41] tool. The graphs are displayed in the Scalable Vector Graphics (SVG) format, allowing the user to interact with the graph by issuing queries by clicking the nodes, thus allowing further analyzation of portions of correlation graph. For example, the user can examine raw alerts behind a summarized alert, IDS signatures that trigger them, the payload, and other relevant information. In addition, to increase navigation ease of the correlations, I implemented an additional interface method. This method presents graphs as list of textual records; each record represents a graph's sink node, *e.g.* $(\mathit{compromised}(H1), \mathit{timerange})$; the record can be clicked and checked for its supporting hypothesis.

2.3.2 Evaluation

The correlation engine was tested on a number of publicly available datasets[91] and on the CIS departmental network. Construction of the reasoning model was conducted separately from the evaluation and without any knowledge regarding specifics of the data sets. The objective of the testing was to ensure that the correlation engine was able to identify different types of attacks and generate attack scenarios. Evaluation on the departmental network analyzes data from various sources such as Snort IDS or black-list logs from CIS department's computer clusters. Snort and SnIPS run on a dedicated Ubuntu server running a Linux kernel version 2.6.32 with 16 GB of RAM on an eight-core Intel Xeon processor of CPU speed 3.16 GHz.

In this section, results from testing the correlation engine on Symantec's Worldwide Intelligence Network Environment (WINE) dataset[14] are presented.

Testing on Wine Dataset

WINE, a platform for repeatable experimental research, is a collection of more than 75 million machines and records occurrences of all known host- and network-based attacks. The following results are scenarios derived from running the correlation engine on WINE Anti-virus (AV) and IPS telemetries. Each scenario graph spans an entire day and supports a compromised machine as a scenario conclusion. This conclusion is represented as a sink node in the graph, *e.g.*, Node 5 in Figure 2.10. All supporting hypotheses (nodes) point toward the compromised node, *e.g.*, Nodes 3 and 4 in Figure 2.10. Each hypothesis is supported by aggregated alerts from wine AV/IPS telemeters, *e.g.*, Nodes 1 and 2 in Figure 2.10. Rectangular nodes represented by AV threats or IPS signatures that trigger alerts. Therefore, a scenario graph can be traced or understood in a bottom-up manner, starting from the compromised machine and tracing back to supporting nodes until the reader reaches aggregated alerts that represent evidence for the scenario. These records are results of blocked attacks by the AV or IPS, meaning that the dataset is dominated by attack records. However, this experiment shows the ability of the SnIPS correlation engine to generate the attack scenario without previous knowledge.

The First Scenario The following scenario supports the conclusion that machine ip_2 is compromised (Node 5, Figure 2.10). In Node 3, ip_1 sent exploit as maliciously crafted Acrobat PDF files to the host ip_2[11]. This event was followed by $Trojan.Bredolab$ compromised machine activity with the same attacker, *i.e.* ip_1 in Node 4[6]. These two events support the

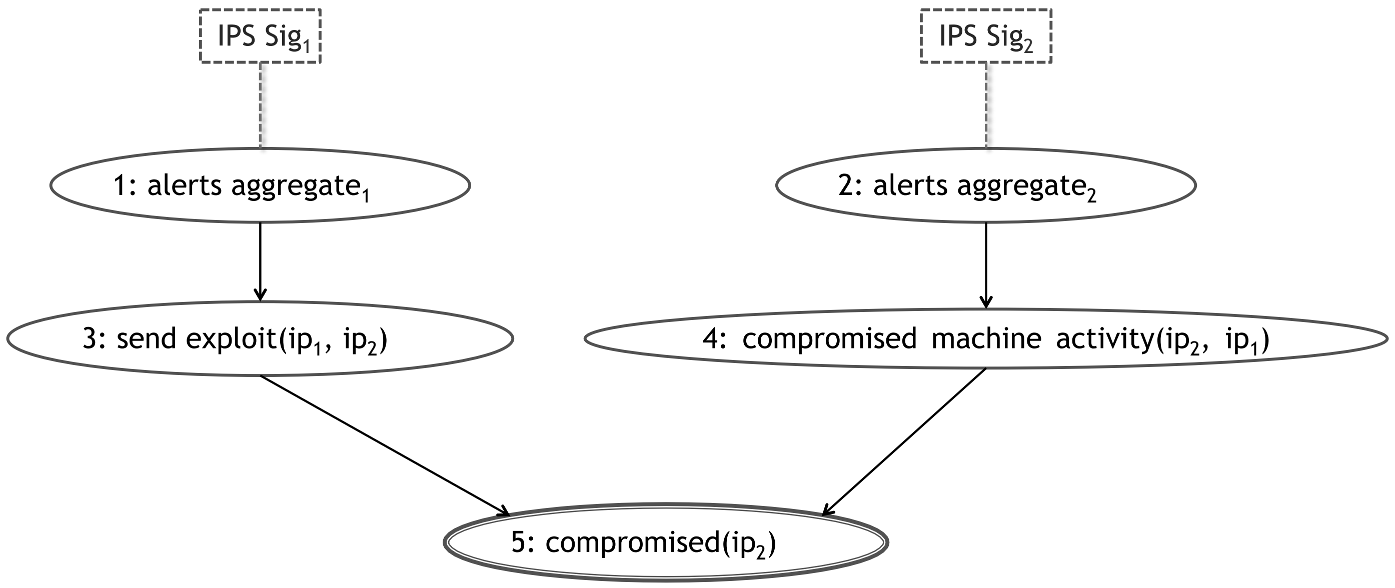


Figure 2.10: *Automatically generated correlation graph from testing the correlation engine on WINE dataset*

hypothesis that the host is compromised, *i.e.*, *compromised*(ip_1). This incident shows that the attacker succeeded in compromising the machine with *Trojan.Bredolab* even though the attack by crafted PDF files had been stopped by the AV. To confirm this scenario, I investigated the attacker ip_1 reputation and found that ip_1 address is marked as malicious by most IP/URL reputation websites[3,5,9]; Symantec lab confirmed the possibility of this scenario, too.

The Second Scenario The following scenario supports the conclusion that machine ip_2 is compromised (Node 7, Figure 2.11). In Node 4, ip_1 sent an exploit as *Trojan.Mebroot* to the host ip_2[13]. Later, the host showed symptoms of the *Backdoor.Tidserv* compromised machine activity (Node 6)[7]. This activity was a HTTPS request which is part of backdoor activity with ip_3. Research has shown that some forms of *Trojan.Mebroot* exhibit similar in behavior to *Backdoor.Tidserv* malware, thus confirming that these two malwares share some of the code segments. This similarity allows *Trojan.Mebroot* to pretend to be *Backdoor.Tidserv* from the IPS/AV perspective[4]. Therefore, the source of compromised activity is a result of the *Trojan.Mebroot* infection that triggered the *Backdoor.Tidserv*

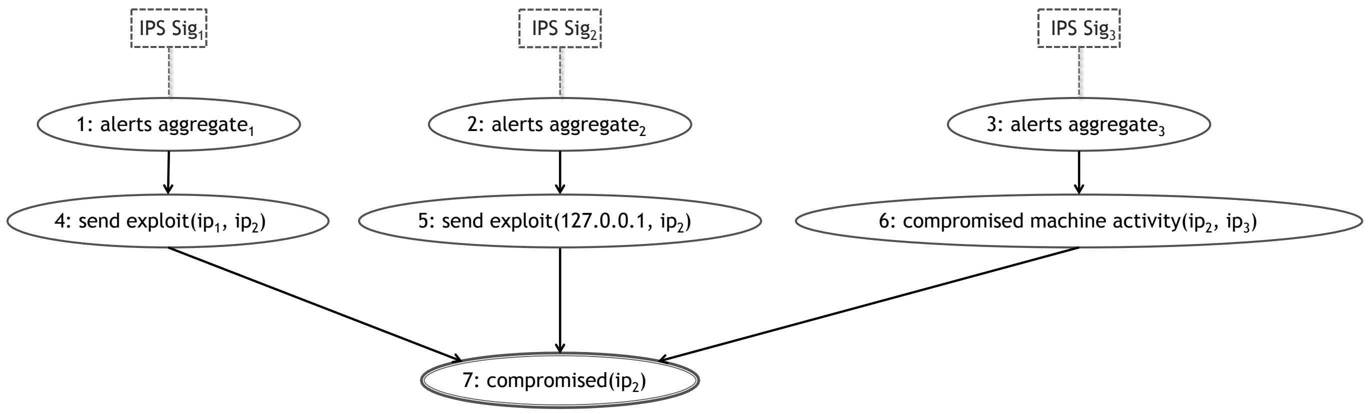


Figure 2.11: *Automatically generated correlation graph from testing the correlation engine on WINE dataset*

alert (IPS Sig 3). In Node 5, 127.0.0.1 (localhost) sent exploit as Fake AV Websites attack the victim machine[1]. The reason for this strange behavior is that the compromised machine may have had a running web service and this service had been used for the Fake AV Websites attacks as a result of infection. In addition, this behavior is common among machines infected by this type of malware. As confirmation of this scenario, I found that ip_1 and ip_3 are marked as malicious by most IP/URL reputation websites[3,5,9]; Symantec lab confirmed the possibility of this scenario, too.

The Third Scenario The following scenario supports the result that the machine ip_2 is compromised (Node 8, Figure 2.12). In Node 4, ip_1 attempted to exploit the host ip_2 through malicious JavaScript code which, when executed, downloads other exploits that can compromise the host[12]. Nodes 5 and 6 show that the host exhibited *Backdoor.Tidserv* compromised machine activity with ip_3 and ip_4[8]. The malicious JavaScript code exploited and downloaded the *Backdoor.Tidserv* that compromised the machine. This graph also has a correlation with an AV event suspected to be a *Backdoor.Tidserv* infection identified by the AV[7].

To confirm this scenario, I found that ip_3 and ip_4 are malicious IPs belonging to the same subnet; ip_1 is also marked as malicious by most IP/URL reputation websites[3,5,9]; Symantec lab confirmed the possibility of this scenario, too.

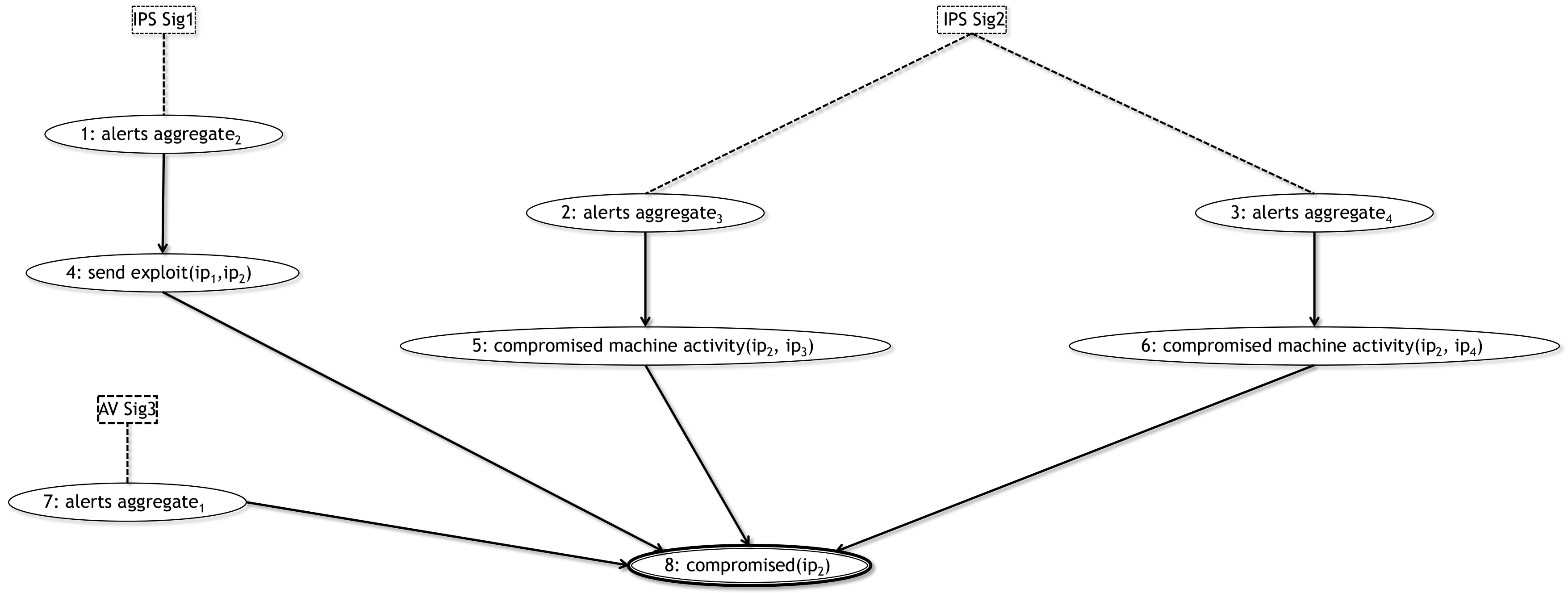


Figure 2.12: *Automatically generated correlation graph from testing the correlation engine on WINE dataset*

Chapter 3

Using Dempster-Shafer Theory to Reduce Uncertainty in Intrusion Analysis

Current IDS systems do not distinguish alarms that are highly likely to be true from alarms that have only a small chance of being true. This leads to a decision dilemma between true and false alarms. When each suspected attack is treated as a hypothesis that may or may not be valid, an effective approach to deal with false positives is to quantify uncertainty in the hypotheses ascribed to IDS alerts by correlating multiple observations relevant to each alert. A list of intrusion hypotheses sorted by confidence and annotated by evidential support for each hypothesis would allow a human analyst to more easily decide which hypotheses are worth further investigation. Most network intrusions involve more than one action. If observations from multiple events are related, a true successful attack will likely contain multiple pieces of corroborating evidence, thus increasing certainty of the attack hypothesis. Similarly, a false positive in one sensor is likely to have less corroborating evidence; thus the particular attack hypothesis can have a low score and can be safely ignored.

DST, one theory that handles uncertainty, is a generalization of the probability theory because of its ability of quantifying unknown[85]. This property makes this theory a superior in situations when the system is in total ignorance. DST also provides a tool to combine multiple evidences flowing from multiple sources.

In this chapter, my approach of applying DST in Intrusion Analysis is elaborated. A brief background on Dempster-Shafer theory is provided in Section 3.1. The extended Dempster-Shafer model and its application to intrusion analysis are described in Section 3.2. Experimental evaluation of the approach is discussed in Section 4.3.

3.1 Dempster-Shafer Theory

DST can be illustrated by an example that shows the difference between probability theory and DST. In this example, if a person tosses a coin with an *unknown bias*, traditional probability assigns 50% for Head and 50% for Tail by the principle of indifference, which states that all states of unknown probability must be assigned equal probability, *i.e.*, uniformly distributed. DST, on the other hand, handles this event by assigning 0% belief to $\{Head\}$ and $\{Tail\}$ and assigning 100% belief to the *set* of $\{Head, Tail\}$, meaning "either Head or Tail". DST thus relaxes the assumption of indifference and does not "force" a number choice when no basis is present on which to assign the number. In general, the DST approach allows for three kinds of answers: *Yes, No, or Don't know.* The last option allows ignorance, thus significantly affecting evidential reasoning.[37,47,80,81,85]

DST provides the following tools:

1. Basic probability assignment function (BPA)
2. Belief function (Bel)
3. Plausibility function (Pl)
4. Translation
5. Rule of combination

3.1.1 Basic Probability Assignment

Before defining *bpa*, frame of discernment must be defined as follows:

Definition 3. Frame of discernment (θ) *is a set of disjoint hypotheses of interest,* e.g., *the set of* {*attack, no-attack*} *or* {*Head, Tail* }.

The *basic probability assignment*, (*bpa*) function, also known as the *mass distribution function*, distributes the belief over the *power set* of the frame of discernment and is defined as follows:

Definition 4. *Let θ be a frame of discernment and m is the bpa function if the following hold:*

$$m : 2^{\theta} \rightarrow [0, 1] \tag{3.1}$$

m is called the basic probability assignment (or mass distribution) on θ.

BPA has the following properties:

Definition 5. *Let θ be the frame discernment, $x \subseteq \theta$, and m is the basic probability assignment then,*

$$\forall x \subseteq \theta, \quad m(x) \geq 0 \tag{3.2}$$

$$m(\{\})\ or\ m(\phi) = 0 \tag{3.3}$$

$$\sum_{x \subseteq \theta} m(x) = 1 \tag{3.4}$$

From the above definition and properties, *bpa* function is distinguished from probability measures by the following:

- It is not required that $m(\theta) = 1$.
- No required relationship exists between $m(x)$ and $m(\bar{x})$.
- $m(x) + m(\bar{x})$ does not have to be 1.

3.1.2 Belief Function

Belief function, or credibility function measures the total belief of all-possible subsets of a hypothesis. Belief function can be defined as follows:

Definition 6. *Let θ be the frame of discernment, m is the basic probability assignment function, and for any $x \subseteq 2^{\theta}$. The belief function is defined as*

$$Bel(x) = \sum_{y \subseteq x} m(y) \tag{3.5}$$

Belief function has the following properties:

$$Bel(\{\}) \text{ or } Bel(\phi) = 0 \tag{3.6}$$

$$Bel(\theta) = 1 \tag{3.7}$$

Belief function and *bpa* are mapping from 2^{θ} to $[0, 1]$. The intuition behind the belief function is that it shows how much confidence is present in a set x in frame of discernment; this confidence is supported by the sum of the weights of multiple evidences. The evidences are all possible subsets of the question of interest x that can support the belief. For example, $m(\{$*attack*, *no-attack*$\})$ is a measure of uncertainty supported by evidence, meaning that we is not sure whether or not there is an attack. On the other hand, *Bel*$(\{$*attack*, *no-attack*$\})$ measures the confidence that *attack* or *no-attack* is true, which must be 1. Belief function is considered the lower bound of support to the hypothesis.

3.1.3 Plausibility Function

Plausibility is another measure, which is not used in my application, but for the sake of completeness, I introduce it in this section. Plausibility function shows the upper bound of how much confidence is present in a set (hypothesis) (x) in the frame of discernment.

Definition 7. *Let θ be the frame discernment, m the basic probability, then for each $x \subseteq 2^{\theta}$, then:*

$$Pl(x) = \sum_{y \cap x \neq \phi} m(y) \tag{3.8}$$

Plausibility function has the following properties:

$$Pl(\{\}) = 0 \tag{3.9}$$

$$Pl(\theta) = 1 \tag{3.10}$$

Belief and Plausibility functions may be viewed as lower and upper bounds on probabilities, respectively, where the actual probability is contained in the interval described by these two non-additive continuous bounds, as follows;

$$Bel(x) \leq P(x) \leq Pl(x). \tag{3.11}$$

Thus, if $Bel(x) = Pl(x)$, the probability measure is uniquely determined, corresponding to classical probability, as shown in the following:

$$Bel(x) = P(x) = Pl(x) \tag{3.12}$$

Belief and Plausibility can be derived from each other. For example, Plausibility can be derived from Belief in the following:

$$Pl(x) = 1 - Bel(\bar{x}) \tag{3.13}$$

where $\bar{x}$ is the classical complement of x. The following formula helps obtain $Bel(\bar{x})$:

$$Bel(\bar{x}) = \sum_{y \subseteq \bar{x}} m(y) = \sum_{y \cap x = \phi} m(y) \tag{3.14}$$

3.1.4 Translation

A crucial aspect of DST is the *translation* process, which Shafer[85] discussed implicitly. Translation process, or *compatibility relation*, is a strength of DST, in which belief functions bases its degrees for one question on the beliefs of a related question. In contrast, Bayesian Networks require probabilities for each question of interest. The intuition of *Translation* can be clarified using the following example from Shafer[85]. Suppose a person has to answer the following question: "Are the streets outside slippery?" For this question frame of discernment, $S = \{yes, no\}$ is created. Suppose *Fred* is a source of information, thus another frame of discernment for Fred's characteristics must be created, which is $F = \{truthful, careless\}$. Suppose Fred said "Streets outside are slippery". The problem is that Fred's statement does not match the question hypothesis S. Thus, two sets S and F exist and the information must be propagated from F to S.

Figure 3.1 provides an example of Translation using intrusion analysis context. E is an IDS alert that indicates a hypothesis H stating "machine ip_1 may be maliciously probing ip_2." Since no IDS sensor is perfect, *i.e.*, does not have 100% accuracy, therefore, its alerts cannot be fully trusted. The frame of discernment of E is {trustworthy, non-trustworthy}. Let us say that, overall, a person has 90% belief in the alert's trustworthiness, *i.e.*, with "90% chance" the sensor works reliably and an alert corresponds to the fact. With 10% chance, the sensor is not reliable, meaning the alert *has nothing to do with* H. Note the difference, traditional probability assigns 10% chance to being H false. When the sensor fires, *bpa* is assigned to {trustworthy} being 90%, and 10% is assigned to {non-trustworthy}. For the hypothesis H, the frame of discernment is {*true*, *false*}, meaning the machine is maliciously probing or not. Table 3.1 illustrates the translation from E to H.

The translation process happens through a *compatibility relation* that specifies which elements in H's frame of discernment are compatible with E's elements. Since the sensor triggers an alert, "trustworthy" is only compatible with *true*element. "Non-trustworthy", on the other hand, is compatible with *true* and *false*elements, meaning when a person does

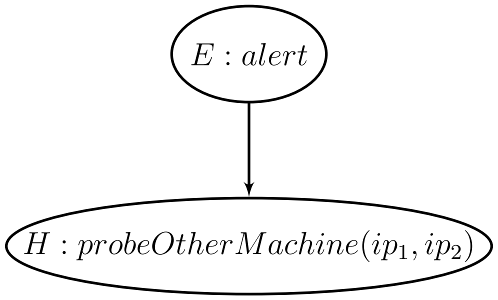


Figure 3.1: *Translation of beliefs*

E			H	
element	value	maps to	element	value
{trustworthy}	0.9	$\rightarrow$	{*true*}	0.9
{non-trustworthy}	0.1	$\rightarrow$	{*true*,*false*}	0.1

Table 3.1: *Example of the translation process in DST*

not trust the alert, the best it can said about whether ip_1 is maliciously probing ip_2 is either *true* or *false*, in other words, "unknown."

3.1.5 Rule of Combination

The goal of combination is to fuse the evidence of a hypothesis from multiple *independent* sources and calculate an overall belief for the hypothesis. Figure 3.2 illustrates this notion, where E_1, E_2 are two alerts triggered by independent IDS sensors[1]. Both alerts could indicate that machine ip_1 is maliciously probing ip_2, which is the hypothesis H. Suppose for each alert a person has 90% belief of its trustworthiness. Suppose that both alerts were fired. How much belief does the person have in H? First, E_1 and E_2's frames of discernment must be translated to H as conducted in Section 3.1.4. Let us name the translated frames of discernment H_1 and H_2. Based on Table 3.1 in Section 3.1.4, the mass functions for H_1 and H_2 are: $m(\{true\}) = 0.9$ and $m(\theta) = 0.1$. All possible combinations between H_1 and H_2 are shown in Table 3.2.

Since the two sensors' operation characteristics are assumed to be independent, the mass

[1] "Independent" means that the two sensors operate on completely unrelated features to determine attack possibilities.

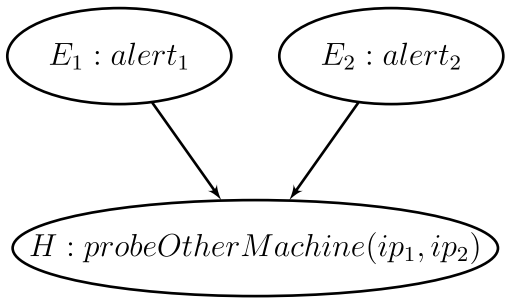


Figure 3.2: *Combination of beliefs*

(h_1,h_2)	*product*	$h = h_1 \cap h_2$
(*true*,*true*)	0.81	{*true*}
(*true*, θ)	0.09	{*true*}
(θ, *true*)	0.09	{*true*}
(θ, θ)	0.01	θ

Table 3.2: *Combination table*

values for H_1 and H_2 can be multiplied in each of the four rows and the numbers for the same resulting subset in H can be added, producing the combined mass function for H. In general, the following rule is known as the *Dempster rule of combination.*

$$m_{1,2}(h) = \frac{1}{1-K} \cdot \sum_{h_1 \cap h_2 = h} m_1(h_1) \cdot m_2(h_2), \text{when } h \neq \{\} \tag{3.15}$$

$$\text{where } K = \sum_{h_1 \cap h_2 = \{\}} m_1(h_1) \cdot m_2(h_2) \tag{3.16}$$

$$m_{1,2}(\{\}) = 0 \tag{3.17}$$

Where h_i is a subset of H_i and h is a subset of H. $1 - K$ is a normalization factor that is a measure of the *conflict* between the two alerts, equivalent to the measure of cases of empty intersection between the H_i's. In this example, no conflict exists ($K = 0$).

The intuition behind the rule of combination in the context of IDS is that when two independent sensors stat the same fact, one belief in the fact will increase. Intersection between H_1 and H_2 leads to possibly reduced uncertainty. Using above formula, the following can be calculated:

$$Bel(\{true\}) = m_{1,2}(\{true\}) = 0.81 + 0.09 + 0.09 = 0.99$$

The normalization factor completely ignores conflict and attributes any mass associated with conflicting masses to the empty set. Therefore, the rule of combination can produce counter-intuitive results[105], and, the reader can find a long line of research about extending the rule of combination to cope with conflicting masses[80]. In my dissertation, I apply this rule monotonically meaning that no conflicts are present, *i.e.*, $K = 0$. The multiplication in Formula 3.15 is only valid when the two evidence sources are independent, which is rare in practice and especially in intrusion analysis since many alerts are generated by the same or related sensors. In the next section, my approach of the customized DST to account for non-independent evidence sources is introduced in order to be correctly applied in intrusion analysis.

3.2 Applying Dempster-Shafer Theory in Intrusion Analysis

DST is uniquely advantageous because it handles uncertainty in intrusion analysis, namely, the lack of need to specify prior probabilities of all events, the ability to quantify the unknown or ignorance, and the ability to combine beliefs from multiple sources of evidence[81,82]. The key question that I attempt to answer in this section is *in a correlation structure, how should a hypothesis's possibility of being true based on the reasoning structure in which it is derived and the quality of supporting evidence be calculated?* Therefore, I represent the lack of knowledge to capture IDS sensor quality, which is often imprecisely described. I present a customization for Dempster's rule of combination that addresses the dependency issue related to application in intrusion analysis. I implemented my method on top of the correlation engine in Chapter 2, so a numeric confidence score can be calculated for a given correlation scenario in order to prioritize.

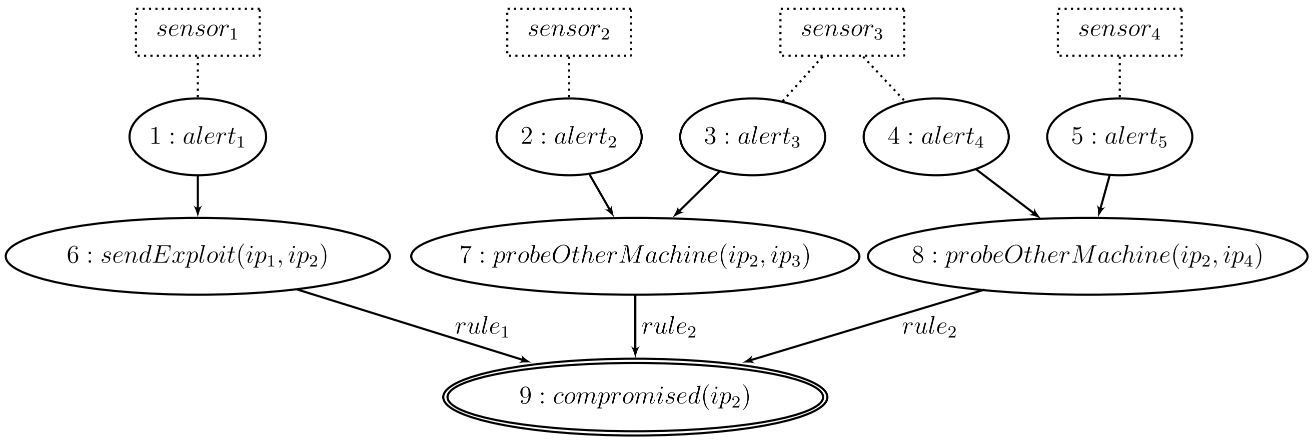


Figure 3.3: *Automatically generated correlation graph from the correlation engine*

3.2.1 Metrics for Sensor Quality

The common standard in measuring IDS sensors quality is false positive and negative rates. These metrics or probabilities may be obtained in other applications by applying the frequentest or objective approach. However, in cyber-security, specifically IDS, obtaining these metrics can be difficult or impossible because this problem is categorized as Subjective uncertainty (Section 1.4), which presents difficulties in obtaining parameters for the system of interest. For example, lack of knowledge exists regarding how to obtain the prior probability of an attack or the probability of not triggering the sensor in the presence of the attack because of the polymorphic behavior of cyber-attacks. Therefore, the ability of DST to quantify the "unknown" is a desirable feature in cyber-security applications. In addition, the nature of *unknown* matches naturally with how humans interpret IDS alerts. For example, when an alert is fired, the security analyst will have a degree (approximately 10%) of belief or confidence that an attack is occurring. On the other hand, a security analyst suppose not to have 90% belief or confidence that the attack is *not* occurring. Observing an alert should raise security analyst confidence that the attack is present, or reduce uncertainty. Adopting the simple *true* and *false* case in order to capture information provided by an alert requires security analyst to know the prior probability of attack or intrusion $Pr[I]$, which is hard to obtain, as shown in the following Bayes' formula:

sensor$_1$'s alert			**sendExploit(ip$_1$,ip$_2$)**	
element	bpa	maps to	element	bpa
{trustworthy}	0.1	$\rightarrow$	{*true*}	0.1
{non-trustworthy}	0.9	$\rightarrow$	{*true*,*false*}	0.9

Table 3.3: *Example of using the sensor's quality metric*

$$Pr[I|A] = \frac{Pr[A|I] \cdot Pr[I]}{Pr[A|I] \cdot Pr[I] + Pr[A|\neg I] \cdot Pr[\neg I]}$$

where A: IDS fires and I: intrusion occurs.

Utilization of DST allows 0.1 belief to be assigned to "attack"({*true*}), 0 belief to "no-attack" ({*false*}), and 0.9 to "unknown" ({*true*, *false*}). This is an intuitive quantitative interpretation and does not provide any belief for "no-attack". *Just because the sensor is not trustworthy, does not mean an attack is not going on. Attack may still occur that is completely outside the scope of the sensor's detection.*

DST method only requires a single metric δ to characterize sensor quality. Therefore, belief calculation begins by assigning this metric to first hypothesis and then propagating the belief to final hypothesis, *i.e.*, frame of interest. To concretely clarify this concept, let us consider the following example:

Example 3.2.1. *Suppose we need to calculate belief in the hypothesis of sendExploit(ip_1, ip_2) in Node 6 of Figure 3.3. We start by using the sensor quality metric δ that corresponds to how much confidence or trustworthiness we have in sensor$_1'$s alerts. Let us say that $\delta = 0.1$ in this example; thus, we construct Table 3.3 to show translation between the two facts.*

In this application, δ is viewed as a metric solely dependent on sensor trustworthiness.

IDS signatures often come with ad-hoc natural-language descriptions that indicate signature quality in terms of how likely the triggered alerts will be false positives. For example,

Measures	maps to	Metrics
unlikely	$\rightarrow$	0.01
possible	$\rightarrow$	0.33
likely	$\rightarrow$	0.66
probable	$\rightarrow$	0.99

Table 3.4: *Example of mapping discrete certainty tags from SnIPS to quantitative sensor quality metrics*

SnIPS (Section 2.1) uses descriptions in Snort signatures documentation to *estimate* a certainty tag for alerts generated by a signature. In practice, such *belief tags* could be explicitly provided by the signature writer. These tags are utilized to map quantitative quality metrics in SnIPS' deployment in CIS departmental network, as shown in Table 3.4.

The mapping is fixed, but different numeric levels can be chosen to assign to a Snort signature, resulting in different *bpa's* assigned to alert trustworthiness which depends on operating environment. According to *Ou, et al.*[67], the intuition for these tags is that humans typically cannot distinguish small differences in numerical parameters, so a few discrete levels are sufficient to express the various beliefs ascribed to an alert.

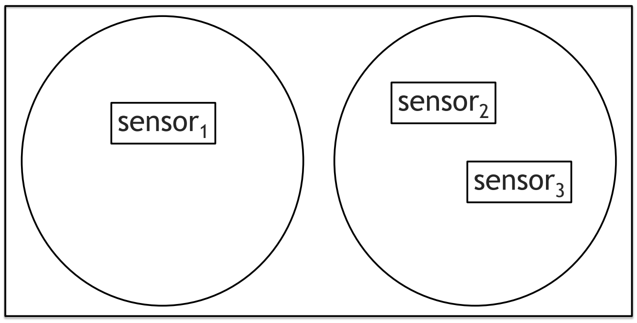
Another benefit of using this model of sensor quality is that no conflict exists among alerts. When security analyst does not trust an alert, he/she says *I Don't know whether the hypothesis is true*, rather than assert that the hypothesis is false, which does not contradict the fact that the user may trust another alert, from which derives the same hypothesis being true. Therefore, this application always has $K = 0$ in the combination rule, Formula 3.15.

3.2.2 Extending Dempster's Rule of Combination

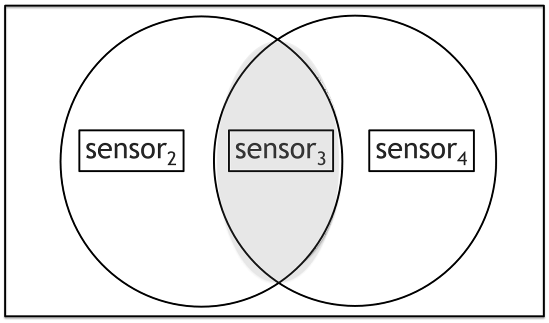
Attack scenarios established by events correlation could provide an elevated belief that an attack is occurring, since multiple pieces of evidence support the same conclusion, *e.g.*, $compromised(ip_2)$ in Figure 3.3. A key question is whether these pieces of evidence come from independent sources. Dempster's rule of combination has a strong assumption about the independence of the mass functions $m_1(h_1)$ and $m_2(h_2)$ in Formula 3.15.

I discovered that the dependent nature of evidences cannot be ignored or avoided in this application. I routinely observed multiple alerts in correlation supporting a hypothesis but these alerts were triggered by the same or similar IDS signatures, leading to an unjustified high level of confidence if the standard Dempster's rule of combination is used.

For example, in Figure 3.3, the main hypothesis is Node 9, meaning "whether machine ip_2 is compromised". This hypothesis is supported by three proof traces that are supported by alert Nodes 1 - 5. Node 7 is supported by Node 2 and 3 (as evidence). Belief in Node 7 is a result combination of beliefs of the supporting nodes. Then we need to combine the belief in Nodes 6, 7, and 8 to answer the final question in Node 9. The fact that these nodes have overlapping evidence cannot be ignored. In particular, Nodes 7 and 8 partially rely on alerts triggered by $sensor_3$, so they are not completely independent and the Dempster rule cannot be applied. Figure 3.4 shows Venn-diagram illustrations for independent and non-independent paths of the graph in Figure 3.3.



(a) Independent evidences



(b) Partially Dependent evidences

Figure 3.4: *Venn diagrams showing dependency of paths in correlation graph in Figure 3.3*

Number of approaches in DST literature account for such dependence[38,83–85]; however, I adopted an idea proposed by Shafer[85], which interprets combined *bpa's* as joint probabilities. I consequently developed a set of customized combination formulas to correctly account for the dependence when combining beliefs in the alert correlation graph.

Customized rule of combination

The joint mass function in Dempster's rule of combination is calculated through multiplication (Formula 3.15), which is the source of independence requirement in this rule. For non-independent evidences, the multiplication of masses from two sources is no longer valid[85]. To handle this limitation, the overlapping part between these masses must considered. Therefore, instead of using $m_1(h_1) \cdot m_2(h_2)$ in Formula 3.15, I propose $\psi[h1, h2]$ to denote the joint masses of the two sources. The following new formula combines possibly non-independent evidences in the correlation graph.

$$m_{1,2}(h) = \sum_{h_1 \cap h_2 = h} \psi[h1, h2] \tag{3.18}$$

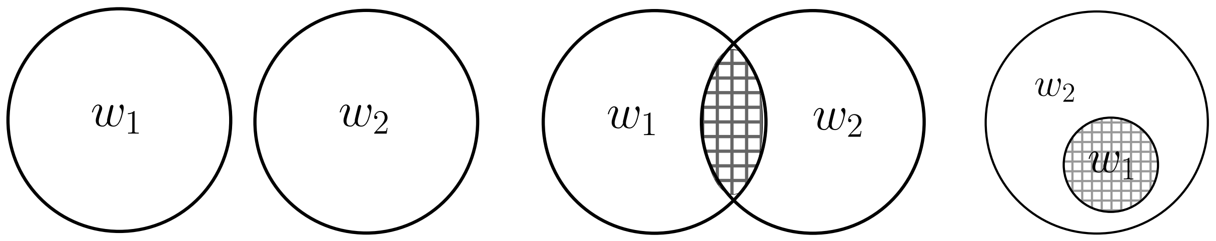
In this application, the only possible h_is are $\{true\}$ (referred to as t) and $\{true, false\}$ (referred to as θ) because of how the *bpa* is assigned to the sensors' frame discernment. As result, only positive correlation exists among IDS alerts because of the absence of conflicts between evidences in the correlation graph, more alerts will not decrease one belief in the existence of attacks. This topic was discussed in Section 3.2.1. The following equations demonstrate how to calculate $\psi[h_1, h_2]$:

$$\psi[t, t] = r_1 \cdot m_1(t) + (1 - r_1) \cdot m_1(t) \cdot m_2(t) \tag{3.19}$$

$$\psi[t, \theta] = (1 - r_1) \cdot m_1(t) \cdot m_2(\theta) \tag{3.20}$$

$$\psi[\theta, t] = (1 - r_2) \cdot m_1(\theta) \cdot m_2(t) \tag{3.21}$$

$$\psi[\theta, \theta] = r_1 \cdot m_2(\theta) + (1 - r_1) \cdot m_1(\theta) \cdot m_2(\theta) \tag{3.22}$$



(a) Independent (b) Non-independent (c) Completely dependent

Figure 3.5: *Venn diagram illustration of evidence dependency*

Semantics of overlapping factors r_i

r_1 and r_2 are *correlation factors* measuring the amount of overlap in the evidence from the two sources. r_1 is the portion of $m_1(t)$ that relies on overlapping evidence from $m_2(t)$. Figure 3.5 shows three cases of dependency between evidence sources. w_i is the statement that $h_i = t$, and the circles represent the IDS signature (sensor) set that triggered the alerts in support of this statement. For example, in Figure 3.5 (a), sensors that support w_1 are completely disjoint from supporting w_2, giving the independent case where $r_1 = r_2 = 0$ and Formulas 3.19 – 3.22 become the standard Dempster's rule of combination. Figure 3.5 (c) illustrates another extreme case where evidence supporting w_1 completely overlaps evidence supporting w_2. In this case, $r_1 = 1$ meaning that when a person believes in w_1, will also believe in w_2. Thus, $\psi[t,t] = m_1(t)$ and $\psi[t,\theta] = 0$. If no evidence can be trusted in w_2, no evidence in w_1 can be trusted, resulting in $\psi[\theta,\theta] = m_2(\theta)$. In general, r_1 will be between 0 and 1, as illustrated in Figure 3.5. Figure 3.4 is a venn diagrams illustration of evidences in Figure 3.3. The first venn diagram identifies, when two graph's paths are independent, and the second diagram demonstrates the partial dependence of the two branches that flows from Node 7 and Node 8 into Node 9.

Since only two non zero-bpa subsets are present, t and θ in each hypothesis's frame of discernment, I utilized w_i to denote $h_i = t$ and $\bar{w_i}$ (negation of w_i) to represent $h_i = \theta$.

Let us define:

$$r_1 = \frac{Pr[w_2|w_1] - Pr[w_2]}{Pr[\bar{w_2}]}, \tag{3.23}$$

$$r_2 = \frac{Pr[w_1|w_2] - Pr[w_1]}{Pr[\bar{w_1}]} \tag{3.24}$$

Let us take r_1 in Formula 3.23 as an example to explain the intuition behind this definition. If one condition of hypothesis h_1 is true, the probability that h_2 is also true is greater than or equal to its absolute probability (shared IDS sensors only give positive correlation). The bigger the difference, the stronger influence h_1 has on h_2. An extreme case is $Pr[w_2|w_1] = 1$, which gives $r_1 = 1$. Both r_1 and r_2 measure dependence between h_1 and h_2.

Theorem 3.2.1.

$$r_2 = \alpha \cdot r_1, \ \ \textit{where } \alpha = \frac{Pr[w_1] \cdot Pr[\bar{w_2}]}{Pr[w_2] \cdot Pr[\bar{w_1}]} \tag{3.25}$$

The proof of the theorem is presented in Appendix A. Let us substitute r_i's definition into Formulas 3.19 – 3.22. Let us also substitute the following definitions:

$$m_i(t) = Pr[w_i] \quad m_i(\theta) = Pr[\bar{w_i}]$$

The following theorem is based on above:

Theorem 3.2.2.

$$\psi[h_1, h_2] = Pr[h_1, h_2]$$

Theorem 3.2.2 can be proved using standard probability theory, which can is also located in Appendix A. This theorem is sound because it is generalization of the joint probability distribution of the trustworthiness of two (potentially) dependent sources.

Estimating overlapping factors The definition of r_i requires knowledge of conditional probabilities $Pr[w_1|w_2]$ and $Pr[w_2|w_1]$, which are unavailable. Therefore, r_i must be estimated in this application. In a given correlation graph, each alert node is associated with a set of IDS signatures that triggered alerts. In this application, these signatures are viewed as independent sensors. For example, Node 7 is supported by alerts from Sensor 2 (Signature 2) and Sensor 3 (Signature 3) in Figure 3.3. In this application, the identity of the sensor triggered an alert is propagated to the supported hypothesis and to other implied hypotheses. Thus, each hypothesis such as h_1 or h_2 is associated with a set of sensors supported by alerts. Each sensor s has a quality metric δ_s, as discussed in Section 3.2.1. Let R_1 and R_2 be the two sensor sets associated with the hypothesis h_1 and h_2 to be combined using Formula 3.18. Let $R = R_1 \cap R_2$. I utilized Formulas 3.26 and 3.27 to estimate the correlation between h_1 and h_2 as follows:

if $\alpha \leq 1$, then

$$r_1 = \frac{\sum\limits_{s \in R} \delta_s}{\sum\limits_{s \in R_1} \delta_s} \quad , \quad r_2 = r_1 \cdot \alpha \tag{3.26}$$

else,

$$r_2 = \frac{\sum_{s \in R} \delta_s}{\sum_{s \in R_2} \delta_s} \quad , \quad r_1 = r_2 \cdot \alpha^{-1} \tag{3.27}$$

where α is defined in (3.25) and can be computed as

$$\alpha = \frac{m_1(t) \cdot (1 - m_2(t))}{m_2(t) \cdot (1 - m_1(t))}$$

I gauged the correlation between two sources by dividing the overlapping evidence's weight by total evidence weight of one branch. Since r_1 and r_2 are related, if a factor is estimated, the other can be computed using α. The above estimation ensures that r_1 and r_2 are within $[0, 1]$. In addition, for both extreme cases in Figure 3.5, the estimation provides accurate results (0 for independence and 1 for complete dependence).

Suppose R_1 and R_2 are two sets of IDS sensors supporting evidences w_1 and w_2, respectively, thus, the following cases are possible:

$$\begin{aligned} &R_1 \cap R_2 = \phi && \text{where,} \quad \phi = \{\}, \text{ then} \quad && r_1 = r_2 = 0 \\ &R_1 \cap R_2 = R && \text{where,} \quad R \neq \phi, \text{ then} \quad && r_1, r_2 \in]0, 1[\\ &R_1 \cap R_2 = R_i && \text{where,} \quad R_i \neq \phi \textit{ and } i \in \{1, 2\}, \text{ then} \quad && r_1 = 1 \textit{ or } r_2 = 1 \end{aligned}$$

3.2.3 Using Translation in Alerts Correlation

This section presents the application of Translation process in the alert correlation system. In Figure 3.3, *bpa* values in alert Nodes 1-5 are translated to their corresponding hypotheses using the method introduced in Section 3.1.4. For example, belief in Node 6 ($\textit{sendExploit}(ip_1, ip_2)$) is translated to Node 9 ($\textit{compromised}(ip_2)$). When an exploit is sent from one machine to another, the target machine will most likely be compromised, but this compromise is not guaranteed because exploits often fail. Therefore, *bpa* of the $\{\textit{true}\}$ in

SendExploit(*any,dest*)		maps to	**Compromised(*dest*)**
{true}	{success}	→	{true}
	{failed}	→	{true,false}
{true,false}	{succ,failed,false}	→	{true,false}

Table 3.5: *Translating sendExploit to compromised:* any *stands for any IP address, and* dest *stands for the destination IP*

sendExploit cannot directly mapped to {true} in *compromised*. One solution is to split the *bpa* based on the predicate's semantics. Thus, the {*true*} element of *sendExploit* is split into two parts corresponding to two possible outcomes of an exploit: success or failure. The weight of each portion is pre-set in the current implementation, but it can also be dynamically adjusted based on exploit difficulty or vulnerability condition on the target host. The success portion is translated to the {*true*} case of *compromised*, and the failure part is translated to the θ case of *compromised*. The assumption is that the failed exploit to a machine X does *not* mean that machine X is not compromised. X could be compromised through other means. This is the rationale behind the translation relation shown in Table 3.5 in which *any* means any IP address and *dest* stands for destination IP.

Translation tables such as Table 3.5 are built corresponding to SnIPS' Internal Model in Section 2.1. Therefore, for every derivation that SnIPS makes between hypotheses, the system knows how to translate *bpa* from the reason to the conclusion. Finally, With the exception of the translation in Table 3.5, no other translations involve splitting *bpa's* weight.

ProbOtherMachine(*source,any*)	maps to	**Compromised(*source*)**
{true}	→	{true}
{true,false}	→	{true,false}

Table 3.6: *Translating probing to compromised:* any *stands for any IP address, and* source *stands for the source IP*

Table 3.6 shows the translation from the fact that a machine is maliciously probing to the fact that the machine is compromised. If a machine performs malicious activities, it must be compromised. Therefore, the {*true*} portion of the *bpa* on *probeOtherMachine* is completely translated to the $\{true\}$ portion of *compromised*.

SendExploit(*source,any*)		maps to	**Compromised(*source*)**
{true}	{succeed}	→	{true}
	{failed}	→	{true}
{true,false}	{succ,failed,false}	→	{true,false}

Table 3.7: *Translating from sendExploit to compromised in the backward direction:* any *stands for any IP address, and* source *stands for the source IP*

Similarly, Table 3.7 shows the translation from *sendExploit* by the source machine to the fact that the *source* machine is compromised. If a machine performs malicious actions, such as sending exploit, thus is compromised.

I used the above translation tables in the current implementation, but the notion of translation is not limited to these tables. The semantics and syntax of these tables are related to SnIPS' Internal Model, which is customizable by design. For example, a new internal rule and translation table can be introduce about probing that is followed by sending exploit. In this section, I presented all necessary preliminaries for my extended approach using DST. Next, I present algorithms for applying this theory.

3.2.4 Belief Calculation Algorithm

This section elaborates on the formal algorithm for applying the customized DST approach. The correlation engine typically generates a list of non-connected alerts-correlation-graph segments such as the one shown in Figure 3.3. Calculation begins by reading this list of graph segments as input. For a given graph segment, the belief of supporting evidences is propagated to the sink nodes, and then the graph list is sorted in descending order by the belief value of the sink nodes. The ranked list is presented to the security analyst , increasing focus on highest-confidence scenarios.

The main algorithm is *DsCorr* (Algorithm 2). This function takes *GraphList*, which is a set of correlation graphs. It calls *ComputeGraphBelief* on each graph segment in the input set and returns a set of graph segments sorted by belief of the sink node (node of final

Algorithm 2: *Ranking graph segments by belief value*

Require: *GraphList* is a list of graph segments.
Ensure: *RankedGraphList* is sorted graph list in descending order by the belief value of the sink nodes.

```
1: function DsCorr(GraphList)
2:     for each Graph in GraphList do
3:         ComputeGraphBelief(Graph) (Algorithm 3)
4:     end for
5:     RankedGraphList ← SortGraphSetbyBelief(GraphList)
6:     return RankedGraphList
7: end function
```

conclusion from the correlation) in descending order. If a segment has more than one sink node, the highest belief value among all sink nodes is taken.

Algorithm 3 takes a graph and computes the belief of every node by propagating evidence from the source nodes (*i.e.*, alert nodes) to the sink node(s) by using breadth-first search algorithm. The algorithm calculates a belief value for each node and returns the highest belief value of the sink node(s). If the correlation graph is cyclic, another algorithm converts it to an acyclic graph by using depth-first search to remove all back edges, yielding a directed acyclic graph (DAG). Then the algorithm starts with source nodes that have *no* parents and insert them in a queue (*ProcessingQueue*). As long as the queue is not empty, the algorithm continues to remove a node, computes its belief value, marks the node as visited. The algorithm also checks for unvisited child nodes with fully visited parents, thus if this condition is met, that child is also placed in the processing queue. This process continues until the algorithm reaches a *sink* node. The algorithm returns the highest belief value of the sink nodes in the graph.

Algorithm 4 takes a node and returns an updated node with belief value and supporting IDS signatures. Three cases must be considered for the node:

1. The node is a source node. *AssignBpaValues* method computes basic probability

Algorithm 3: *Compute the belief for each node in the graph segment*

Require: *Graph* has an array of nodes.
Ensure: record the highest belief value of sink nodes.

```
1: function ComputeGraphBelief(Graph)
2:     MakeAcyclic(Graph)
3:     ProcessingQueue ← all the source nodes
4:     while (ProcessingQueue is not empty) do
5:         Node ← ProcessingQueue.RemoveHead
6:         Node ← ComputeNodeBelief(Node) (Algorithm 4)
7:         Node.visited ← true
8:         for each C in Node.Children do
9:             if all C's parents are marked visited AND C.visited = false then
10:                ProcessingQueue ← C
11:            end if
12:        end for
13:    end while
14:    record the highest belief value of sink nodes.
15: end function
```

Algorithm 4: *Compute the belief value of a node*

Require: *Node* a *Graph* node.
Ensure: *Node′* is updated node with belief value and sensors list.

```
1: function ComputeNodeBelief(Node)
2:     if Node.Parents has no parents then
3:         AssignBpaValues(Node)
4:     else if Node has one parent P then
5:         Node.belief ← Translate(P)
6:         Node.sigSet ← P.sigSet
7:     else if Node has multiple parents PS then
8:         Node.belief ← Combine(PS)
9:         Node.sigSet ← union of PS.sigSet
10:    end if
11:    return Node
12: end function
```

assignment based on method in Section 3.2.1. This case applies to alert nodes, *e.g.*, Node 1-5 in Figure 3.3.

2. The node has only one parent node. The translation function propagates the belief from the parent to the current node, as discussed in Section 3.2.3.

3. The node has multiple parents. The parents are implicitly translated into the current-node semantic and the combination process is applied.

Algorithm 5: *The Combination Process*

Require: *ParentsList* is a list of parents nodes.
Ensure: *Belief* is the combined belief value.

```
1: function COMBINE($ParentsList$)
2:     for each P in $ParentsList$ do
3:         $TranslatedParentsList \leftarrow$ TRANSLATE($P$)
4:     end for
5:     FILLPRIORITYQUEUE($TranslatedParentsList$)
6:     $Belief \leftarrow$COMBINEPAIRWISE($PriorityQueue$)
7:     return $Belief$
8: end function
```

Algorithm 5 reads a node's parents list and returns the combined belief value. The algorithm begins by translating the parents into the current-node semantic. In the case of more than two parents to combine, Node 9 in Figure 3.3, they are combined in pairwise manner. The customized combination Formula 3.18 is associative because it computes joint probability of hypotheses, thus it can handle evidence in any order. In this algorithm, I ordered branches dynamically in descending order by belief values (using priority queue) to fuse the highest-quality information first. IDS-signature identifications are propagated through the graph to estimate the correlation factor using Formulas 3.26 and 3.27.

Complexity The complexity for this algorithm is *O(N.G)*, where N is the number of graph segments in $GraphList$ in Algorithm 2, and G is the size of the graph segment. In the worst case, the generated graph is quadratic in the number of IP addresses in the alerts.

3.2.5 Illustrative Example

In this section an illustrative example demonstrates belief calculation process using the correlation graph segment in Figure 3.3. In this figure, sensor nodes are present for the purpose of explanation, but they are not part of the graph in practice. The steps of belief calculation are shown below.

Step 1: Compute belief values for source nodes.

The algorithm starts from alert Nodes 1 – 5 which are each associated with the triggering sensor (IDS signature). Let the sensor's quality metric be δ based on the definition in Section 3.2.1. Suppose $\delta = 0.6$ for Node 1. The following steps explain the process:

(a) Create frame of discernment, $\theta = \{true, false\}$

(b) Create the power set out of (θ), $2^\theta = \{\{\}, \{true\}, \{false\}, \{true, false\}\}$

(c) Distribute the measures on the 2^θ, using *bpa*. In Table 3.8, the empty set receives *Zero*, $(m(\{true\}))$ takes the measures (*e.g., 0.6*) from Node 1. *Zero* is then assigned to $(m(\{false\}))$, because no supporting evidence exists for this element. Finally, $(m(\{true, false\}))$ receives the remaining mass.

In these steps, I implicitly map the trustworthiness frame of discernment of the alerts to {true, false} frame of discernment.

Step 2: Propagate belief values using Translation.

In this process, I translate semantics of the source node to the destination node using a set of translation tables, as explained in Section 3.2.3. For example, to translate from Node 6 to Node 9, I consulted Table 3.5.

bpa	**values**
$m(\{\})$	0
$m(\{true\})$	0.6
$m(\{false\})$	0
$m(\{true, false\})$	0.4

Table 3.8: *Bpa table for Node 1 in Figure 3.3*

$sendExploit(ip_1, ip_2)$				$compromised(ip_2)$	
element	value	element	value	element	value
{true}	0.6	{success}	0.36	{true}	0.36
		{failure}	0.24	{true,false}	0.24
{true,false}	0.4	{success,failure,false}	0.40	{true,false}	0.40

Table 3.9: *Example of translating sendExploit to compromised*

Table 3.9 shows the translation process with computed values. The left side of the table has the $\{true\}$ element for *sendExploit*has belief value of 0.6. This is split into {success} and {failure} using a pre-set 3 : 2 ratio, assuming that if an exploit is sent to a machine, this machine "likely" is compromised. Finally, I added up translated weights for each subset and obtained 0.36 as the belief for $compromised(ip_2)$ being *true*.

Step 3: Combine belief values.

Combination is applied when multiple derivation paths lead to a single node. Node 9 is as an example, which has three pieces of evidences flowing into it from Node 6–8[2]. First, the nodes' belief values based on their perspective semantics are translated into the *bpa* of Node 9's semantics $compromised(ip_2)$. The algorithm sorts the three branches based on translated belief value and combines top two branches. Then the combined branches are similarly combined with the rest of the branches. Let us assume that Nodes 7 and 8 are the first pair to be combined; Node 7's belief value after translation is 0.68 and Node 8 's value is 0.6.

[2]I use the word "evidence" broadly since 6–8 are actually hypotheses with supporting evidence.

To combine, I used the customized combination formula 3.18–3.22. To estimate the overlapping factors r_1 and r_2, I used Formulas 3.26 and 3.27, resulting in $m_1(t)$=0.68 and $m_2(t)$=0.6; $R_1 = \{sensor_2, sensor_3\}$ and $R_2 = \{sensor_3, sensor_4\}$. Quality metrics for the sensors are $\delta_{sensor_2} = 0.2$, $\delta_{sensor_3} = 0.6$, and $\delta_{sensor_4} = 0.01$. Therefore, I have the following:

$$\alpha = 1.42$$

Since $\alpha > 1$, the following is used:

$$r_2 = \frac{\sum\limits_{s \in R} \delta_s}{\sum\limits_{s \in R_2} \delta_s} \quad , \quad r_1 = r_2 \cdot \alpha^{-1} \quad \text{then,}$$

$$r_2 = \frac{0.6}{0.61} = 0.98, \quad r_1 = 0.98 \cdot 1.42^{-1} = 0.69$$

(h_1,h_2)	$\psi(h_1, h_2)$	h
$(true,true)$	0.597	$\{true\}$
$(true,\ \theta)$	0.083	$\{true\}$
$(\theta,\ true)$	0.003	$\{true\}$
$(\theta,\ \theta)$	0.317	{true,false}

Table 3.10: *Belief combination example*

Table 3.10 has four tuples for the combination of two evidences. The first element in the tuple is from the first evidence and the second is from the other. For each row, I used the Formulas 3.19-3.22, respectively. For example, I used Formula 3.19 to calculate $(true,true)$ tuple. Finally, by adding the values in the first three rows, the belief is $Bel(\{\text{true}\}) = 0.683$.

The overlapping $sensor_3$ weight was not count in the result. The interested reader can verify if the standard Dempster rule of combination was used, the result is much higher (0.872) since $sensor_3's$ weight is double-counted. Using standard rule produces significant amplification on the graph segment, leading to unjustified higher ranking.

Finally, is to combine this result with the belief from Node 6, which is similarly calculated. The sensor set associated with the combined belief is the union of sensor sets from all branches.

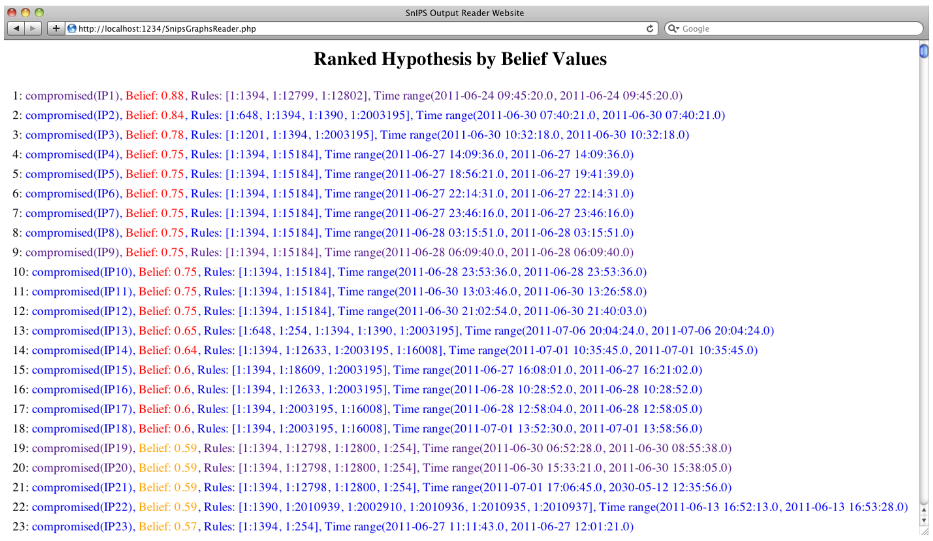
3.3 Implementation and Evaluation

In this section, I elaborate on the implementation and evaluation of the customized DST application algorithm.

3.3.1 Implementation

The implementation of the customized DST algorithm is in Java. The prototype reads the input as a list of correlation graphs from a back-end database. The back-end MySQL database is used by the correlation engine to store output graphs (Section 2.3.1), and the prototype carries out the algorithm described in Section 3.2.4. The program then outputs a list of correlation segments sorted by highest belief values of the sink nodes. This list is stored in the back-end database to be used in later stages of the implementation.

I utilized a web interface visualizer implemented in PHP and HTML. The graphs visualization was generated using the Graph Visualization Software (GraphViz)[41] tool. Furthermore, to increase the ease of correlation navigation, I implemented an additional interface method, which presents the graphs as a list of textual records. Each record represents a graph's sink node, $(compromised(H_1), timerange, beliefvalue)$. Each compromised record can be clicked and checked for its supporting nodes with their calculated belief values. Figure 3.6 shows part of the interface.


SnIPS Output Reader Website
http://localhost:1234/SnipsGraphsReader.php
Google

Ranked Hypothesis by Belief Values

```
1: compromised(IP1), Belief: 0.88, Rules: [1:1394, 1:12799, 1:12802], Time range(2011-06-24 09:45:20.0, 2011-06-24 09:45:20.0)
2: compromised(IP2), Belief: 0.84, Rules: [1:648, 1:1394, 1:1390, 1:2003195], Time range(2011-06-30 07:40:21.0, 2011-06-30 07:40:21.0)
3: compromised(IP3), Belief: 0.78, Rules: [1:1201, 1:1394, 1:2003195], Time range(2011-06-30 10:32:18.0, 2011-06-30 10:32:18.0)
4: compromised(IP4), Belief: 0.75, Rules: [1:1394, 1:15184], Time range(2011-06-27 14:09:36.0, 2011-06-27 14:09:36.0)
5: compromised(IP5), Belief: 0.75, Rules: [1:1394, 1:15184], Time range(2011-06-27 18:56:21.0, 2011-06-27 19:41:39.0)
6: compromised(IP6), Belief: 0.75, Rules: [1:1394, 1:15184], Time range(2011-06-27 22:14:31.0, 2011-06-27 22:14:31.0)
7: compromised(IP7), Belief: 0.75, Rules: [1:1394, 1:15184], Time range(2011-06-27 23:46:16.0, 2011-06-27 23:46:16.0)
8: compromised(IP8), Belief: 0.75, Rules: [1:1394, 1:15184], Time range(2011-06-28 03:15:51.0, 2011-06-28 03:15:51.0)
9: compromised(IP9), Belief: 0.75, Rules: [1:1394, 1:15184], Time range(2011-06-28 06:09:40.0, 2011-06-28 06:09:40.0)
10: compromised(IP10), Belief: 0.75, Rules: [1:1394, 1:15184], Time range(2011-06-28 23:53:36.0, 2011-06-28 23:53:36.0)
11: compromised(IP11), Belief: 0.75, Rules: [1:1394, 1:15184], Time range(2011-06-30 13:03:46.0, 2011-06-30 13:26:58.0)
12: compromised(IP12), Belief: 0.75, Rules: [1:1394, 1:15184], Time range(2011-06-30 21:02:54.0, 2011-06-30 21:40:03.0)
13: compromised(IP13), Belief: 0.65, Rules: [1:648, 1:254, 1:1394, 1:1390, 1:2003195], Time range(2011-07-06 20:04:24.0, 2011-07-06 20:04:24.0)
14: compromised(IP14), Belief: 0.64, Rules: [1:1394, 1:12633, 1:2003195, 1:16008], Time range(2011-07-01 10:35:45.0, 2011-07-01 10:35:45.0)
15: compromised(IP15), Belief: 0.6, Rules: [1:1394, 1:18609, 1:2003195], Time range(2011-06-27 16:08:01.0, 2011-06-27 16:21:02.0)
16: compromised(IP16), Belief: 0.6, Rules: [1:1394, 1:12633, 1:2003195], Time range(2011-06-28 10:28:52.0, 2011-06-28 10:28:52.0)
17: compromised(IP17), Belief: 0.6, Rules: [1:1394, 1:2003195, 1:16008], Time range(2011-06-28 12:58:04.0, 2011-06-28 12:58:05.0)
18: compromised(IP18), Belief: 0.6, Rules: [1:1394, 1:2003195, 1:16008], Time range(2011-07-01 13:52:30.0, 2011-07-01 13:58:56.0)
19: compromised(IP19), Belief: 0.59, Rules: [1:1394, 1:12798, 1:12800, 1:254], Time range(2011-06-30 06:52:28.0, 2011-06-30 08:55:38.0)
20: compromised(IP20), Belief: 0.59, Rules: [1:1394, 1:12798, 1:12800, 1:254], Time range(2011-06-30 15:33:21.0, 2011-06-30 15:38:05.0)
21: compromised(IP21), Belief: 0.59, Rules: [1:1394, 1:12798, 1:12800, 1:254], Time range(2011-07-01 17:06:45.0, 2030-05-12 12:35:56.0)
22: compromised(IP22), Belief: 0.59, Rules: [1:1390, 1:2010939, 1:2002910, 1:2010936, 1:2010935, 1:2010937], Time range(2011-06-13 16:52:13.0, 2011-06-13 16:53:28.0)
23: compromised(IP23), Belief: 0.57, Rules: [1:1394, 1:254], Time range(2011-06-27 11:11:43.0, 2011-06-27 12:01:21.0)
```

Figure 3.6: *Example of textual format user interface*

3.3.2 Evaluation

The prototype was continuously evaluated on the production network throughout my research. The production network consists of approximately 200 servers and workstations, including Windows, Linux, and Mac OS X. The Snort alert collection, correlation, and DST algorithm application were all carried out on a Ubuntu server running a Linux kernel version 2.6.32 with 16GB of RAM on a eight core 3.16GHz Intel Xeon processor.

I found that, although SnIPS can infer a certainty tag for each Snort rule by analyzing the rule's documentation, estimated certainty tags often are not consistent. SnIPS may map one Snort rule to "likely", whereas a similar rule may be mapped to "possible". In such cases, I manually modified certainty tags for rules triggered on the production network based on my understanding of rule descriptions. Snort rules often need to be grouped since they detect similar patterns and cannot be deemed "independent". For example, Snort rules for detecting Network File System (NFS) related network traffic are placed in single group. In this work, these manual adjustments are called "refinements" of system input parameters.

IDS systems often cast into doubt the validity of evaluation effectiveness since the tool may be trained to work well for a specific system. To avoid bias in the refinement process, I refined Snort rule's trustworthiness level based on the generic meaning of the rule without considering system specifics on which the rule is triggered. However, to maintain evaluation integrity, refinement was conducted in a "context-agnostic" manner. After I refined parameters on the production network, I applied the system to two additional datasets. Both datasets have "truth files" which can be used to compare against the ranking provided by DST algorithm. I used the following datasets:

1. Lincoln Lab DARPA intrusion detection evaluation dataset.

2. Predict dataset.

Evaluation Methodology

The objective of evaluation is to test *whether the customized DST is effective in prioritizing IDS alerts correlations.* To that end, I assigned a belief value to each IDS alert that was the highest belief of the supporting hypotheses. This can be calculated from the alert correlation graph through linear traversal. If IDS alerts with high belief values turn out more likely to be true alerts than those with low belief values, it is an indication of effectiveness of the approach.

In order to demonstrate that the application of customized DST helps in prioritization, I compared the performance of the customized DST algorithm to the following alternative methods:

1. Using sensor quality metrics only. In this method, I used sensor's quality metrics assigned to each alert as an alert's belief value.

2. Using maximum sensor's quality metric in a correlation graph as the belief value for all alerts in the graph.

3. Using belief values calculated from standard DST instead of the customized rule.

Each method assigns a belief value to each IDS alert. A threshold value is set and alerts with belief values above the threshold are classified as true alerts; those below the threshold are classified as false alerts.

I used truth files included in the dataset to compare against classification provided by belief values. As the belief-value threshold changed, the classifier obtained different operating points in terms of true positive rate and false positive rate. Therefore, I used Receiver operating characteristic (ROC) curves for the four methods to compare their performance. I also used precision, recall, and sample size for each operating point, in effort analysis. In the effort analysis, I showed the effect of alerts prioritization to reduce the workload of the security analyst.

Results from Predict Dataset

Predict dataset was prepared and released by Skaion Corporation for the Disruptive Technology Office (DTO), a funding agency within the United States Intelligence Community, now called the Intelligence Advanced Research Projects Activity (IARPA). This dataset is a simulation for the Open Source Information System (OSIS) network, which is an unclassified network used by the intelligence community to share sensitive-but-unclassified information. The notional network consists of a central internet-connected backbone, combined with (fictional) intelligence community subnets. Subnets contain a collection of services and clients that run Windows and Linux operating systems. The nature of the background traffic is a mixture of benign packets and failed attacks, and successful attacks are documented as ground truth scenarios.

The dataset has *Postgre* database dumps of Snort alerts. To run the experiment, I first ran SnIPS' correlation engine to generate correlation graphs, which were stored in a back-end database to run the DST module in the future. To facilitate the testing process, I implemented an automatic testing program (in Java) that read the ground truth and correlation graphs to plot ROC, sensitivity, and efforts (prioritization effects) curves. The ground truth was created by going through each attack scenario and checking against the alert database to select alerts that could be verified as true alerts. The remainder of the alerts were declared as false alerts.

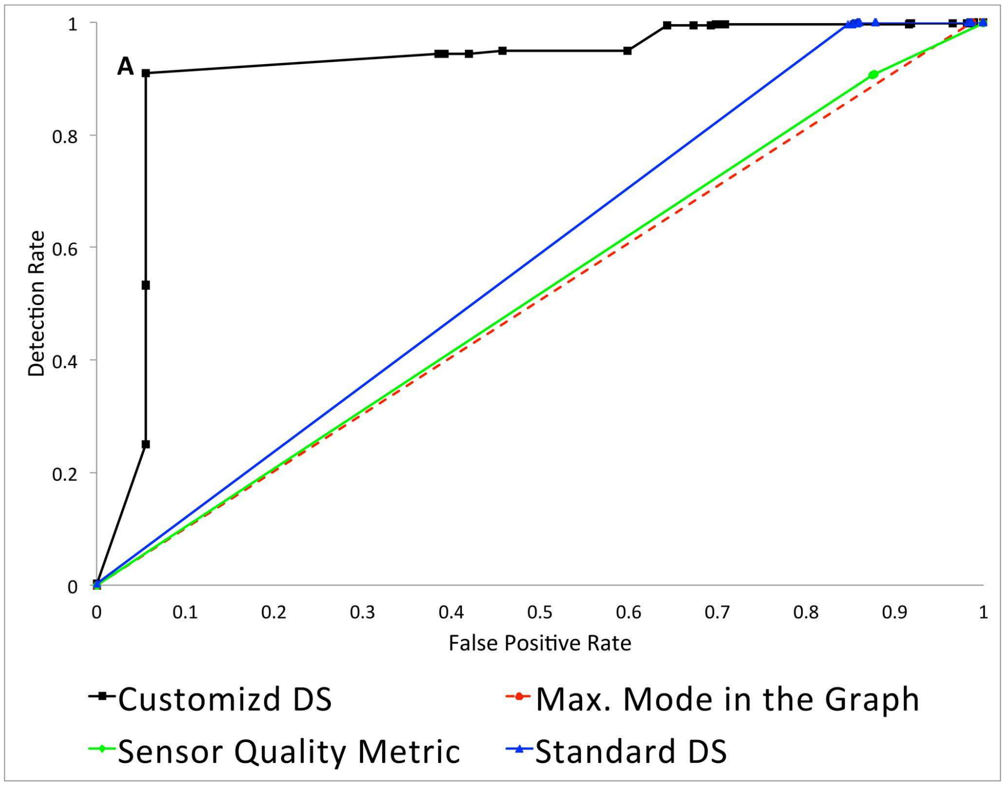


Figure 3.7: *ROC Curves for Predict dataset*

The ROC curve is a standard method to compare performance of IDS systems[18]. It shows the relationship between detection rate (true positive) and false positive rate of a classifier. In general, the steeper and closer to the left-up corner, the better the classifier. A comparison of ROC curves generated for the Predict dataset is shown in Figure 3.7. The curves clearly indicates that the customized DST algorithm outperforms the three alternative methods. The customized DST algorithm produces the most optimal operating point A corresponding to belief threshold 0.87. The calculating formulas for TPR and FPR are shown below for reference.

$$\text{True Positive Rate (TPR)} = \frac{\text{number of true alerts above threshold}}{\text{number of total true alerts}}$$

$$\text{False Positive Rate (FPR)} = \frac{\text{number of false alerts above threshold}}{\text{number of total false alerts}}$$

Prioritization Effect The primary objective of DST application is to use relative belief values to prioritize intrusion analysis. Figure 3.8 shows how precision and recall changes when the threshold decreases from 1 to 0. (Note that 0 in the X axis corresponds to belief 1, and 1 corresponds to belief 0). When the reader starts with alerts with high beliefs, precision is high, meaning the security analyst effort is devoted to useful tasks. When the threshold decreases, cumulative precision decreases, indicating that calculated belief values can be effectively used to prioritize further investigation.

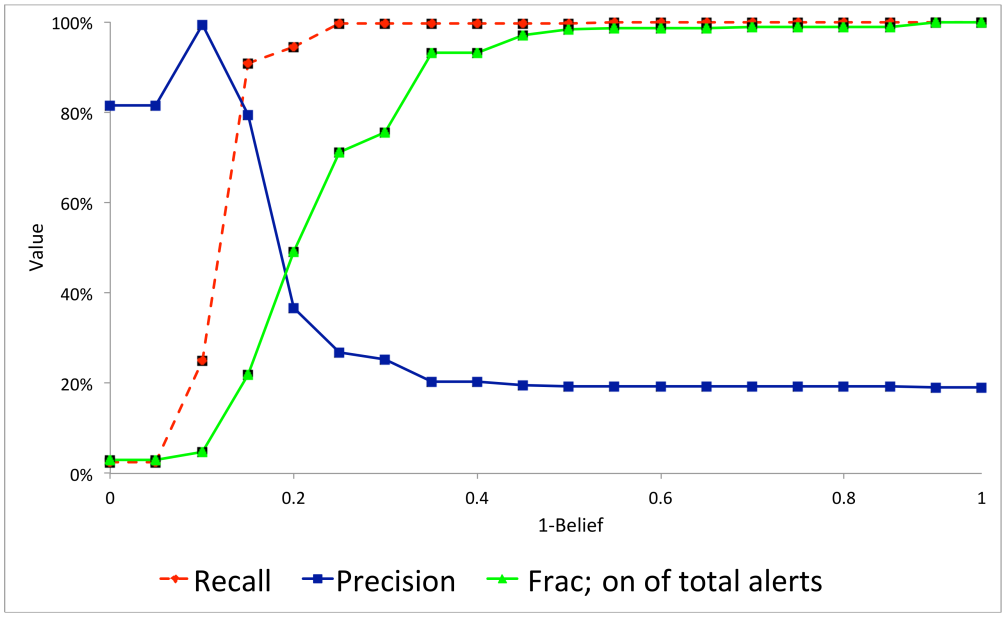


Figure 3.8: *Prioritization effect for Predict dataset*

For example, for 0.87 belief value (0.13 point at the X axis), the percentage of total alerts captured was approximately 21%, and the recall was approximately 90%, meaning that if only alerts set with the highest belief (*e.g.* > 0.87) are analyzed, this set only includes 21%

of all alerts, and covers 90% of all true alerts. Recall and precision calculating formulas are shown below for reference.

$$\text{Precision} = \frac{\text{number of true alerts above threshold}}{\text{number of alerts above threshold}}$$

$$\text{Recall} = \frac{\text{number of true alerts above threshold}}{\text{number of total true alerts}}$$

Sensitivity Analysis I experimented with the effect of variation on the choice of sensor's quality metric values. I varied default mapping in four ways, each of which increased all numeric values by approximately 0.03, *e.g.*, from 0.3 to 0.33. I compared results from all four cases along to the default case using ROC curves (Figure 3.9).

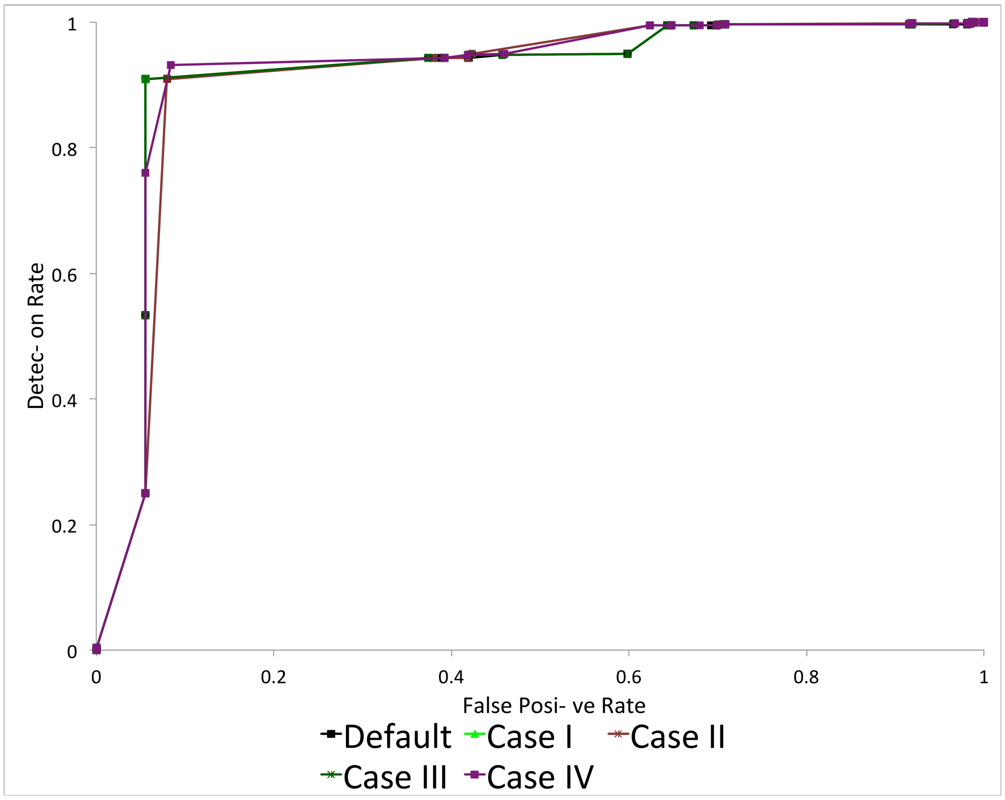


Figure 3.9: *Sensitivity analysis' ROC curves for Predict dataset*

Figure 3.9 shows that the five curves overlap each other, indicating that small change in one direction for certainty tag values has negligible effect classifier performance. However, I found that the system is sensitive if the change in mapping gets to more than 0.15 in one direction. Sensitivity is revealed by the decrease in system performance, resulting in worse ROC.

Results from Lincoln Lab DARPA Dataset

The system was tested on the MIT Lincoln Lab DARPA intrusion detection evaluation dataset. Although the Lincoln Lab dataset has been criticized in the literature[59,60], it is still one of the few usable publicly available datasets for IDS research because it is supported by good documentation of ground truths and the existence of background and attack traffic. The limitation of the LL dataset does not significantly affect the validity of this evaluation for the following reasons:

1. Most problems identified in the LL dataset affect anomaly-based detection[59] in which data for training and testing purposes must be utilized. For example, one issue is that the Time to live (TTL) value of the packets is a decisive feature in classifying attack and non-attack traffic. This example does not affect a signature-based IDS such as Snort, which I used as the underlying alert source.

2. SnIPS' reasoning model was built, from existing Snort rule repositories before the testing occurred. It is calibrated on the departmental network, which is completely unrelated to the LL dataset.

3. The problem in LL dataset's background traffic[60] makes it hard to make claims on the performance of the tested system on real networks, especially since it is a very old dataset. Therefore, I primarily used the dataset to compare performance. The relative performance of various methods is likely not affected as much as the absolute performance since they may all benefit or suffer from specific features of the dataset.

DARPA 1998 and 1999 Training Data To generate correlation graphs, Snort and SnIPS were run on the packet capture (pcap) format of the dataset. Generated correlations were stored in a back-end database to run future experiments. To facilitate the testing process, I implemented an automatic testing program (in Java) that read the ground truth and correlation graphs to plot ROC, sensitivity, and efforts curves. Ground truth was created by going through each attack scenario and checking against the alert database to choose alerts that can be verified as true; the remainder of the alerts were declared as false.

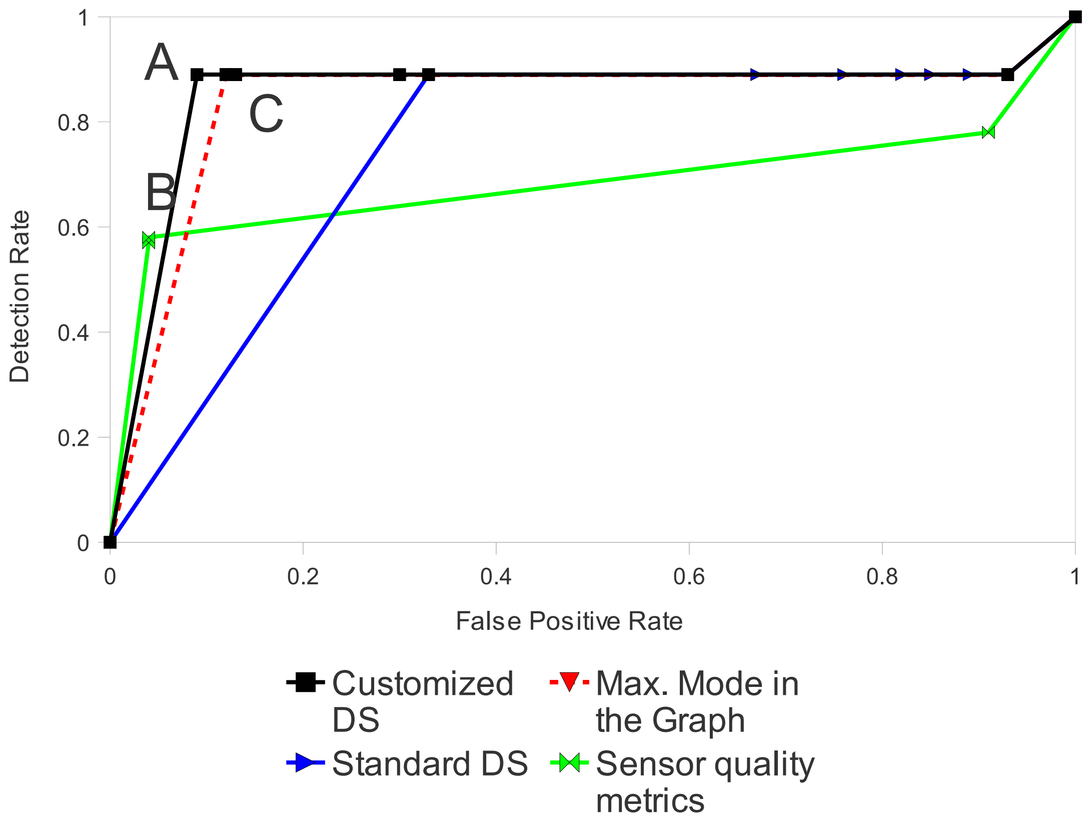


Figure 3.10: *ROC curves for Lincoln Lab 1998 dataset*

A comparison of ROC curves generated for both datasets is shown in Figure 3.10 and 3.11. The curves indicate that the customized DST algorithm outperforms the three alternative methods. Some operating points of the other three methods come close to the customized DST algorithm for LL 98 data, *e.g.*, point B and C. but these points are inferior to LL 99 data. Conversely, the customized DST algorithm consistently produces the most

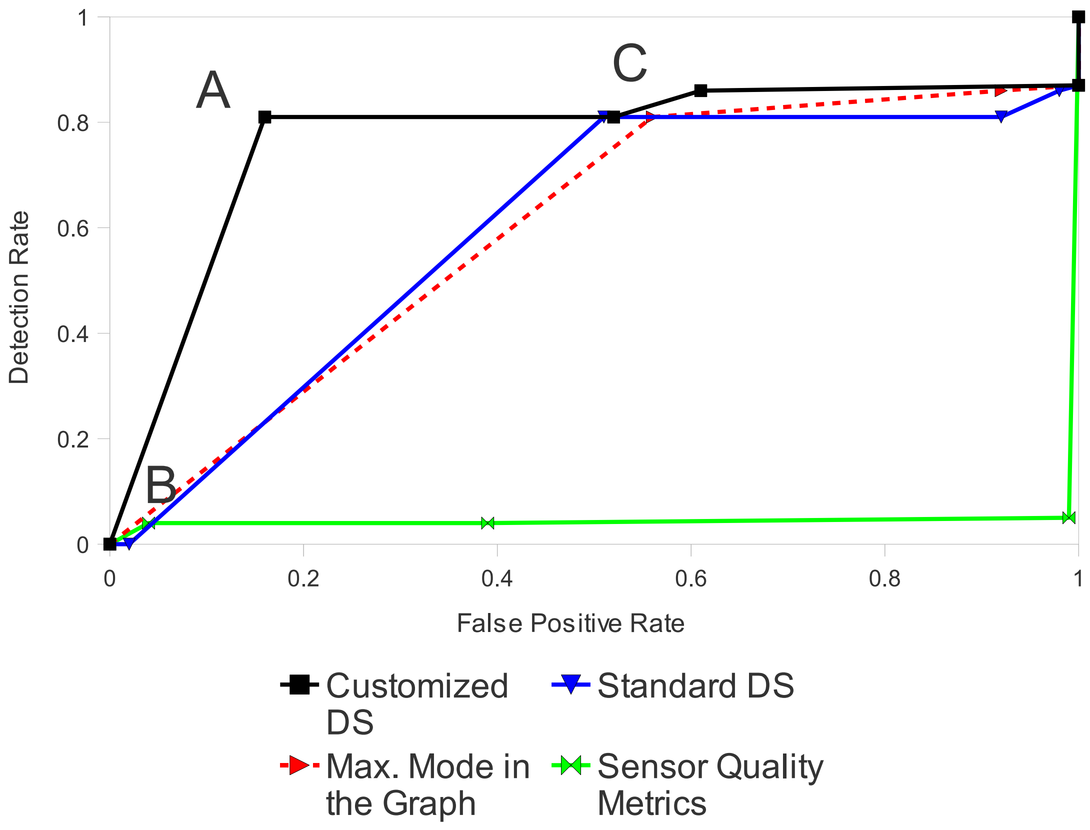


Figure 3.11: *ROC curves for Lincoln Lab 1999 dataset*

optimal operating point for both graphs (point A, corresponding to belief threshold 0.9).

Prioritization effect The primary objective of DST application is to use relative belief values to prioritize intrusion analysis. Figures 3.12 and 3.13 demonstrate how precision and recall changes when the threshold decreases from 1 to 0. (Note that 0 in the X axis corresponds to belief 1, and 1 corresponds to belief 0). When one starts with alerts with high beliefs, the precision is high meaning more of the effort is devoted to useful tasks. When the threshold decreases, the cumulative precision decreases, indicating that calculated belief values can be used effectively to prioritize further investigation.

At the highest belief range (0 point at the X axis), the percentage of total alerts captured was approximately 40% and the recall was approximately 80%. This means that if only alerts slice with the highest belief were analyzed (*e.g.*, > 0.9), it only includes 40% of all alerts,

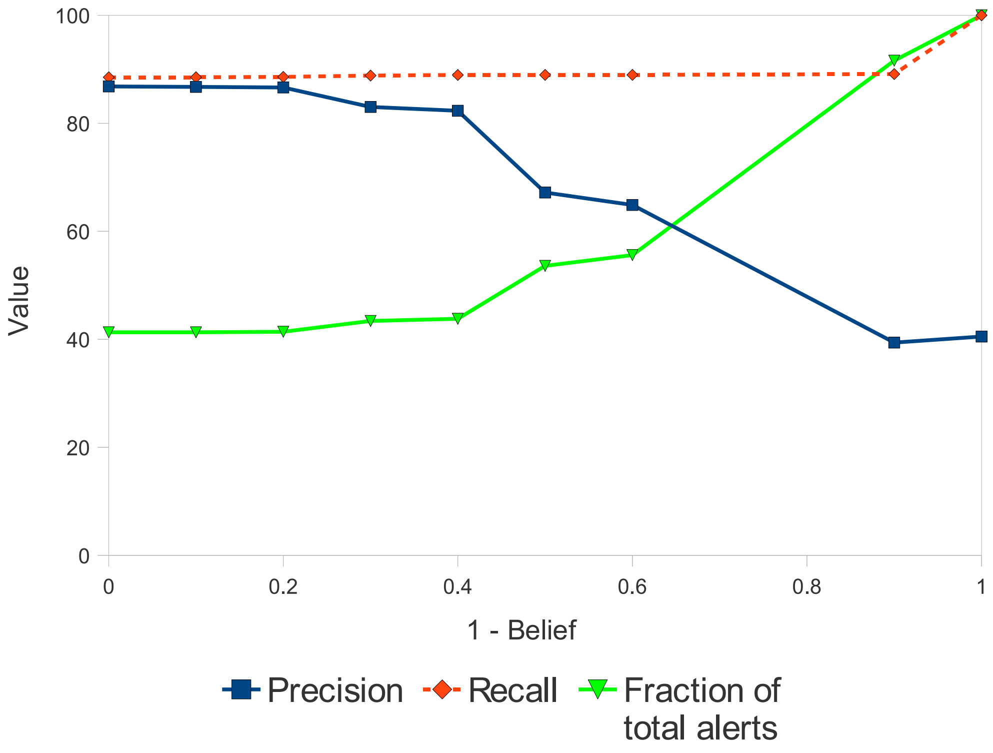


Figure 3.12: *Prioritization effect for Lincoln Lab 1998 dataset*

and covers 80% of all true alerts. The recall curve is flat, meaning that most attacks can be captured using a high threshold value. This is certainly the case for these two datasets, but it also indicates the effectiveness of prioritization provided by the DST method. Thus in the absence of prioritization, security analyst must to analyze twice as many alerts to achieve the same coverage.

Sensitivity Analysis I experimented with the effect of variation on the choice of sensor's quality metric values. I varied the default mapping shown in table 3.4 in four ways, each of which decrease all the numeric values by approximately 0.03, *e.g.*, from 0.33 to 0.3. I compared results from all four cases and the default case in ROC curves for LL 99 data.

Figure 3.14 indicates that the five curves overlap, indicating that small change in one direction for certainty tag values has virtually no effect on classifier performance. However, I found that the system is sensitive if the mapping change gets to more than 0.15 in one

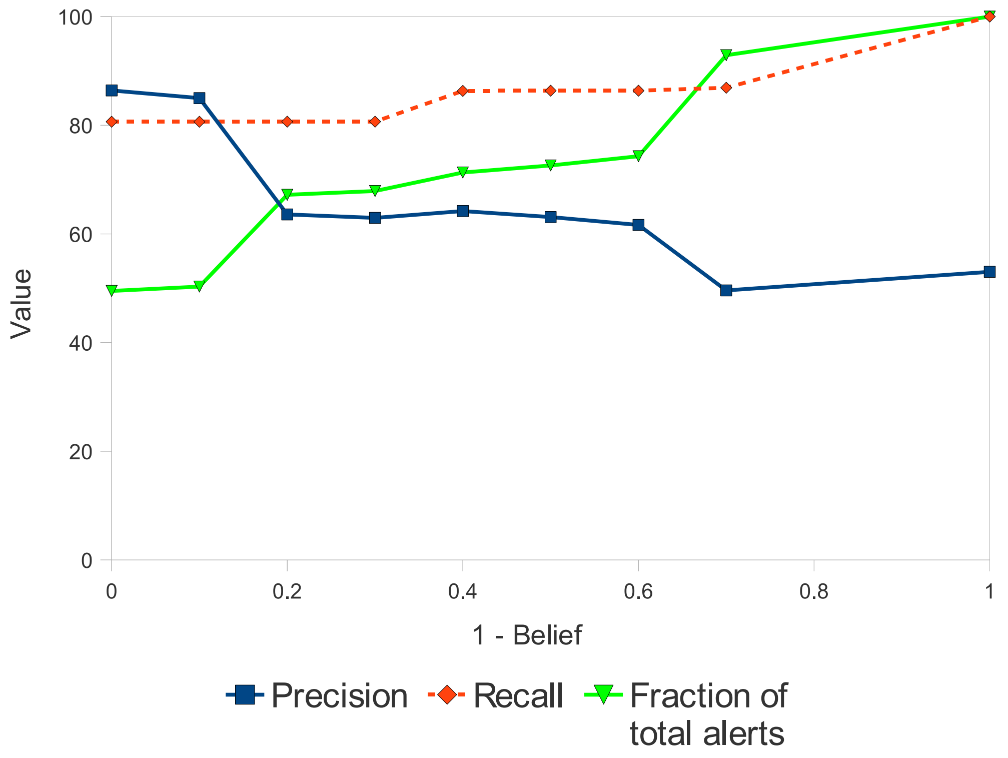


Figure 3.13: *Prioritization effect for Lincoln Lab 1999 dataset*

direction. The sensitivity revealed by decreased system performance, resulting in worse ROC.

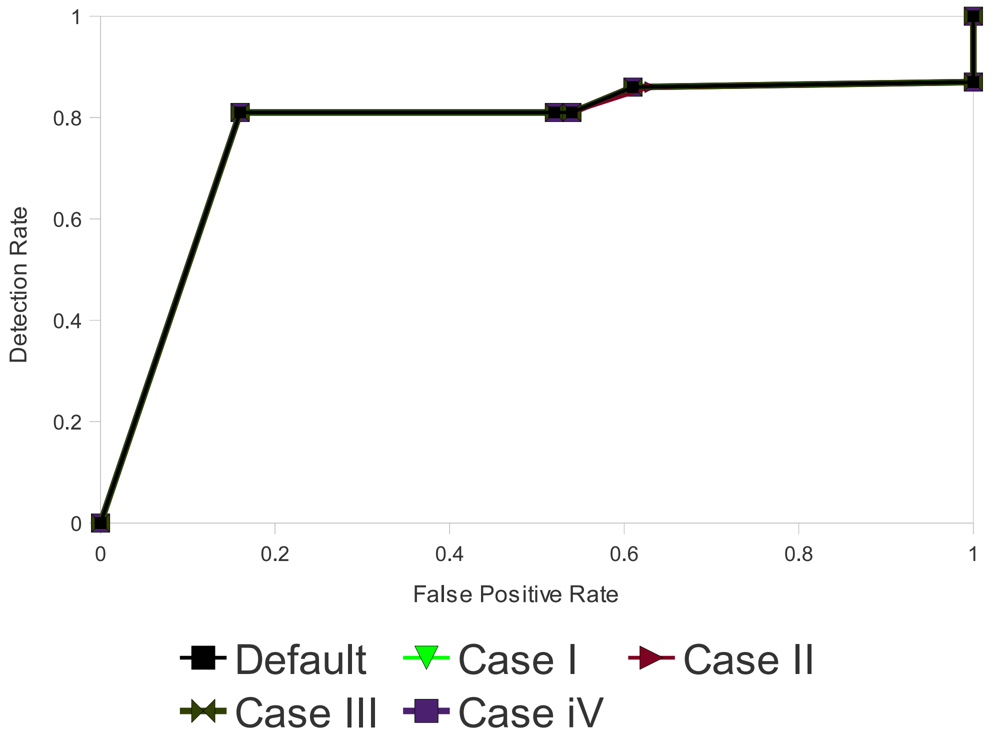


Figure 3.14: *Sensitivity analysis' ROC curves for Lincoln Lab 1999*

Chapter 4

Using Machine Learning to Reduce Uncertainty in Intrusion Analysis

Observation of SnIPS [1] output that runs on the departmental network, revealed that an security analyst must perform repetitive tasks to pruning out false positives in ranked correlations because of inability to feedback the system with security analyst's decisions. In one incident, security analyst found a high-ranking noisy server that turned out to be a benign DNS server. This event continuously appears on the graphs list and the ranking cannot be changed, according to findings from the security analyst. Therefore, I hypothesized that such repetitive tasks can yield (limited) labeled data that can enable the use of a machine learning-based approach to prune output based on security analyst feedback, similar to spam filters that learn from users' past judgment to prune emails[42,44,51]. The goal of this chapter is to classify correlation graphs produced from SnIPS' DST prioritization module into "interesting" and "non-interesting", where "interesting" means that security analyst needs to conduct further analysis on the events. I manually labeled SnIPS' output based on this criterion and built prediction models using supervised and semi-supervised learning approaches. The experiments revealed a number of insightful observations into the pitfalls and challenges of applying machine learning in intrusion analysis. Experimentation results also indicated that semi-supervised learning is a promising approach to practical machine

[1]The correlation engine and customized DST implementation has been combined with SnIPS. SnIPS intrusion analysis framework is currently available as an open source software[54].

learning-based tools that aid security analysts, when a *limited* amount of labeled data is available.

Section 4.1 presents research hypotheses and Section 4.2 discusses the approach for the application of machine learning in intrusion analysis using SnIPS. Section 4.3 presents experimental results and discussion.

4.1 Research Hypothesis

Through empirical study of the output of deployed SnIPS on the departmental network, I found that many correlations are not interesting because they are supported by noisy IDS signatures and the correlation structure does not provide significant boost on the belief in attacks. When the correlation seems to be interesting (potential attack) , additional information currently not captured by SnIPS must often be consulted. In order to use a tool such as SnIPS, the security analyst must first determine whether the output correlation is worthy of further investigation. While it would be ideal if the calculated belief value could determine whether further investigation is warranted, practical experience shows that this is not always the case. Therefore, manual work is still necessary to select interesting output from SnIPS for further analysis. In this chapter I hypothesis the following:

Hypothesis 4.1.1. *The possibility to automate the manual process of determining whether an IDS alert correlation graph is interesting and worthy of further investigation through a machine-learning approach.*

This hypothesis is justified by the fact that a security analyst can determine the usefulness of an IDS alert correlation graph, indicating that features within the graph are predictive to the classification of "interesting" and "non-interesting", where "interesting" means that the graph is worthy of further investigation. In addition, a security analyst must validate the alert correlation graph, implying that, ultimately, the human analyst must make final decision on graph usefulness. This process creates limited amount of labeled data to train a machine-learned classifier. Therefore, a machine learning approach could be feasible

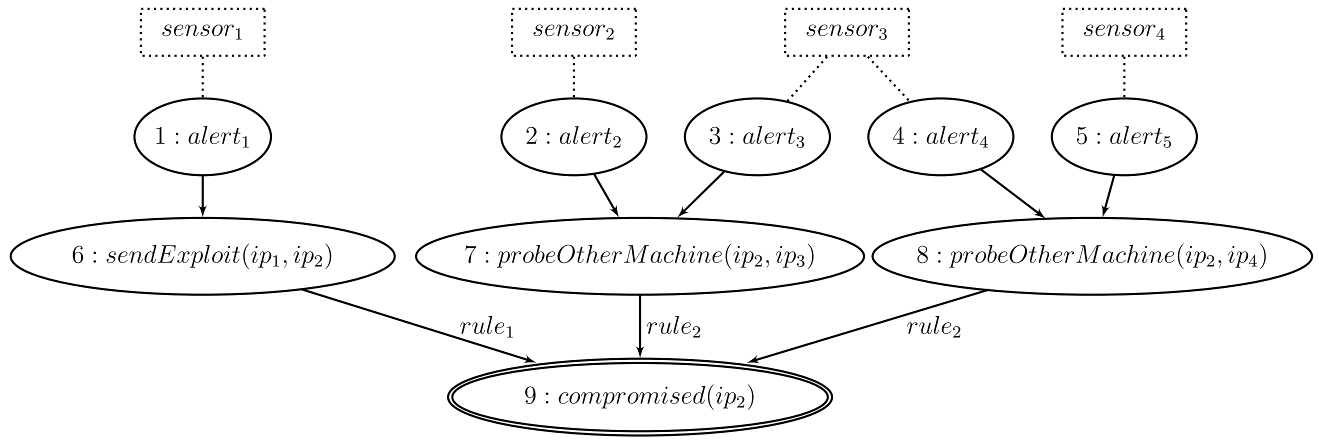


Figure 4.1: *Automatically generated correlation graph from the correlation engine*

for operational use, where a human analyst analyze the output and continuously provides feedback on the classifier's precision in the form of fresh-labeled data.

4.2 Methodology

This work aim to utilize features existing in a correlation graph to build a prediction model to classify whether output correlation is "interesting" or "non-interesting." Each correlation graph has one or more sink nodes representing the correlation's conclusion. The classification task is to determine whether the correlation is worthy of investigation based on features included in the correlation graph, including graph structure, details of supporting evidence, and the belief value of the graph.

4.2.1 Classes

The sink node for the graph is considered the data point for the classification, *e.g.*, Node 9 in Figure 4.1. Using this data point, the correlation graph can be classified into the following two classes:

1. *Interesting:* This class means that the correlation is found to be highly suspicious and worthy of additional in-depth investigation. This class covers a wider area than "true

positive," since the conclusion drawn by the correlation may turn out to be a false positive after further investigation.

2. *Non-interesting:* This class means that the correlation is found to be non-interesting and not worthy of further investigation. Ideally, the conclusion must be false positive; however, in reality, an attack could be true but no sufficient evidence is captured in the correlation so the graph is mistakenly dismissed as "non-interesting." [2] The "non-interesting" class means that no sufficient evidence exists to warrant further investigation of the correlation.

4.2.2 Dataset Construction

I utilized data from the departmental network for training and testing. This is consistent with the envisioned use of the prediction model in which each organization trains its own prediction model based on security analysts' feedback using the correlation tool. Multiple datasets could produce stronger evidence on a prediction model's effectiveness; however, the particular problem addressed in this research requires access to production networks. I fortunately have had supportive local system administrators and was able to conduct this experiment on the CIS network, but practical constraints made obtaining access to multiple production networks infeasible. Even though experimentation was conducted on a single network, the results I obtained provide useful insights into the effectiveness of this approach and how to generally apply machine learning for intrusion analysis, especially, compared to artificially generated traffic. Furthermore, collecting and labeling the dataset provided me with invaluable insights, thereby aiding feature selection process, as described later in this chapter.

Network Setup and Labeling Tool

The departmental network consists of 35 Linux and 11 Windows servers and more than 150 workstations, including Sun, Mac Pro, and PC running Windows and Linux. The

[2]Improving true positive of intrusion analysis is an orthogonal problem of this research.

departmental network consists of three VLANs: main, printer, and thin clients. The main VLAN contains all servers and user machines and is the entry point to the departmental network. It is a giga-bit switched network with a giga-bit uplink to the campus network. A fiber optic cable is attached to Cisco switch of the main VLAN to mirror all traffic including ingress and egress traffic for the departmental network and internal traffic for the main VLAN. Snort IDS system runs on captured traffic, which produces tens of thousands of alerts per day. Snort and SnIPS run on a dedicated Ubuntu server running a Linux kernel version 2.6.32 with $16GB$ of RAM on an eight-core Intel Xeon processor of CPU speed $3.16GHz$.

To facilitate the labeling process, I implemented a web-based interface that allows a security analyst to interact with SnIPS' output and determine on whether it is interesting or non-interesting. Feedback is recorded in a back-end database along with the features extracted from correlation graph and supporting evidence. The web interface is implemented in PHP and HTML, and the graphs visualization was generated using the Graph Visualization Software (GraphViz)[41] tool. Graphs are displayed in Scalable Vector Graphics (SVG) format, allowing user to interact with the graph by issuing queries by clicking the nodes, thus aiding analyzation of portions of the correlation graph. For example, security analyst can examine raw alerts supporting a summarized alert, triggering IDS signatures, payload, and other relevant information. This drill-down feature is useful in the labeling process. In order to increase ease of correlation navigation, I implemented additional interface method, which presents graphs as a list of textual records; each record represents graph's sink node such as $(\mathit{compromised}(H1), \mathit{timerange})$, which can can be drilled down for its supporting nodes.

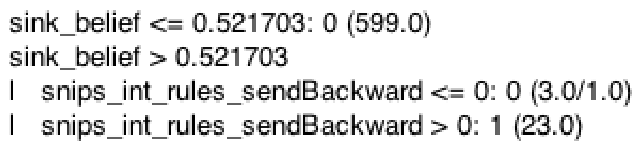
Labeling Process

The labeling process was the most time-consuming part of this research because of the need to obtain sufficient amount of labeled data for training and testing. However, when the tool is deployed and used in operation, the labeled data will be generated "naturally" by security analysts when they analyze correlation graphs. As the tool is used over time, continuously fresh-labeled data will be generated to train the prediction model. Although a tremendous amount of initial work is required for researchers who must be trained as operation analysts and analyze large amounts of data for research experiments, such effort would be part of routine work of security analysts in a deployed setting and virtually no additional cost would be required for labeling.

The ideal research method would prefer that labeling be done by a real (full time) security analyst. However, such professionals are difficult to recruit for research purposes. Research prototypes are not as easy-to-use as mature off-the-shelf products, and security analysts are typically overwhelmed by a variety of tasks with limited time to assist researchers. Therefore, I acted as a security analyst to further examine SnIPS' output in order to understand and label the events, which is a less-than-ideal situation in regard conflicts of interest. To prevent bias, multiple persons should label the data whenever possible.

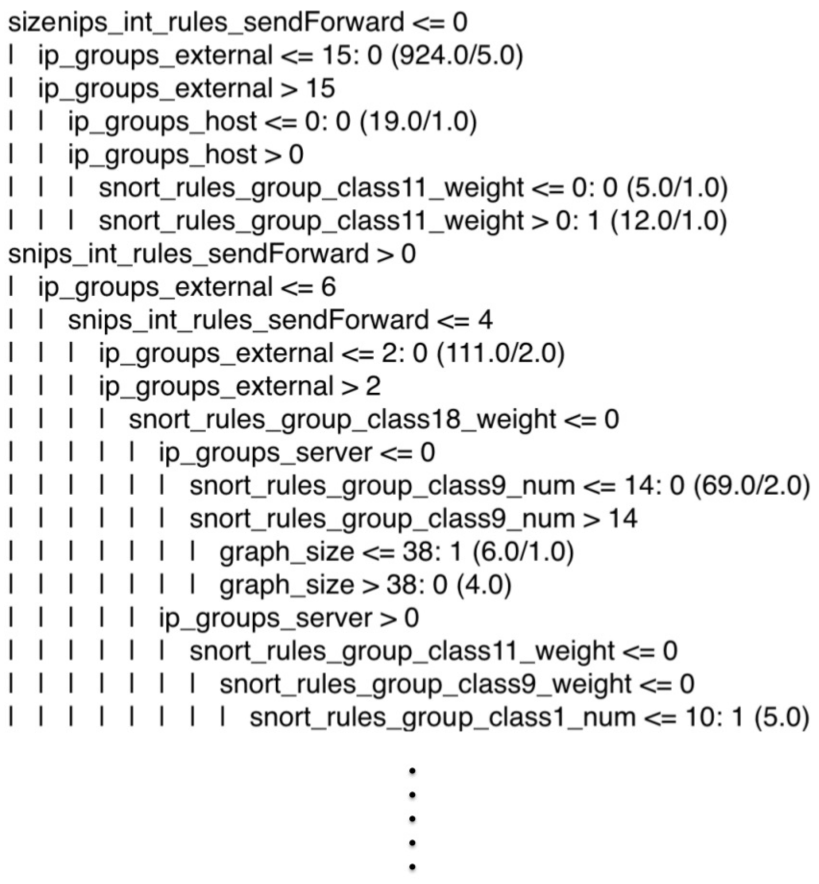
The labeling process went through multiples rounds. The first round, I labeled dataset A consisting of 160 data points. The second round labeled dataset B with approximately 625 data points. Datasets A and B had two issues: insufficient size for experimentation and bias in the labels to the belief values, resulting in forcing the classifier to use only belief for classification; therefore, the results were too good to be true.

To better understand the reason for the unexpectedly good results, I conducted experiments using a decision-tree classifier, which builds a binary tree with leaves representing class labels and branches representing conjunctions of features that lead to those class labels. Figure 4.2 illustrates the overall difference, in terms of decision trees, between datasets B and C. For dataset B, the output decision tree has five nodes with three leaves. The main distinguishing feature is the belief value of 0.52. However, when I ran the same algorithm on dataset C, the result was a tree with 97 nodes and 49 leaves, meaning that the second decision tree used a majority of the features to classify. The reason for this difference is that when I labeled dataset B, the belief values were shown to the user and the belief value



```
sink_belief <= 0.521703: 0 (599.0)
sink_belief > 0.521703
|   snips_int_rules_sendBackward <= 0: 0 (3.0/1.0)
|   snips_int_rules_sendBackward > 0: 1 (23.0)
```

(a) dataset B



```
sizenips_int_rules_sendForward <= 0
|   ip_groups_external <= 15: 0 (924.0/5.0)
|   ip_groups_external > 15
|   |   ip_groups_host <= 0: 0 (19.0/1.0)
|   |   ip_groups_host > 0
|   |   |   snort_rules_group_class11_weight <= 0: 0 (5.0/1.0)
|   |   |   snort_rules_group_class11_weight > 0: 1 (12.0/1.0)
snips_int_rules_sendForward > 0
|   ip_groups_external <= 6
|   |   snips_int_rules_sendForward <= 4
|   |   |   ip_groups_external <= 2: 0 (111.0/2.0)
|   |   |   ip_groups_external > 2
|   |   |   |   snort_rules_group_class18_weight <= 0
|   |   |   |   |   ip_groups_server <= 0
|   |   |   |   |   |   snort_rules_group_class9_num <= 14: 0 (69.0/2.0)
|   |   |   |   |   |   snort_rules_group_class9_num > 14
|   |   |   |   |   |   |   graph_size <= 38: 1 (6.0/1.0)
|   |   |   |   |   |   |   graph_size > 38: 0 (4.0)
|   |   |   |   |   ip_groups_server > 0
|   |   |   |   |   |   snort_rules_group_class11_weight <= 0
|   |   |   |   |   |   |   snort_rules_group_class9_weight <= 0
|   |   |   |   |   |   |   |   snort_rules_group_class1_num <= 10: 1 (5.0)
                              .
                              .
                              .
                              .
                              .
```

(b) dataset C

Figure 4.2: *Comparison between decision trees for B and C datasets*

likely influenced the labeling decision. When I labeled dataset C, the belief values were hidden and the SnIPS output graphs were randomly re-ordered to ensure that the user was not influenced by the belief value. This method helped reduce the bias to the graph belief value or to the graph rank. Dataset C size is 1615 data points, which is moderately sized to conduct the evaluation, as shown in Section 4.3.

Labeling Guidelines To ensure consistency, these guidelines were followed throughout the labeling process. For each correlation graph, I performed the following steps to determine whether a graph is "interesting" or "non-interesting".

1. Check the graph structure to understand and validate the scenario that supports the hypothesis (sink node), *e.g.*, Node 9 in Figure 4.1.

2. Check type of the machines involved in the graph, either internal machines (clients or servers) or external machines. For external machines, I used "whois" service and IP reputation websites, *e.g.*, "Trend Micro site safety center" (`http://global.sitesafety.trendmicro.com`) to get a sense on whether they are benign or potentially malicious.

3. Check open ports on the machines involved in the graph, in order to discover any malicious services on these machines and to verify some Snort alerts.

4. Check Snort alerts' payloads to gain insights into the reason behind each alert. This only provides a limited view however, because Snort only stores the triggering part of the packet for the alert.

5. Check other features such as timestamp, graph size, and Snort-signature categories in the correlation.

Observations on Labeling Process

Throughout the labeling process, multiple correlation graphs marked the departmental DNS server as compromised because of many DNS queries to some blacklisted domains. If a user,in the internal network, wants to access websites in blacklisted domain, network's DNS server sends queries to authoritative name servers for those blacklisted domains. Those name servers are likely to be blacklisted, too. Blacklist IPs were from the Emerging Threat Snort signatures that flag *any* communication with a blacklisted IP. This behavior puts the DNS server at the top of the list as it triggers large number of these alerts, however, little chance exists for the DNS server to be compromised. This observation indicates that contextual information, such as a host's intended functionality, plays important role in understanding security alerts triggered for the host. For an effective learning-based approach, such information must be accounted for in the process of selecting features.

In addition a Snort's *sfPortScan* rule was found to be noisy in the departmental network and was tuned to avoid obvious false positives. For example, one client machine in the departmental network was browsing multiple websites simultaneously. A Snort rule was triggered and an alert was generated because a single machine accessing multiple machines on a single port (80 in this case) was sufficient for the signature to match. These type of rules made generic to capture all types of port scans. However, a fine-grain tuning is needed for this rule and many other rules. This type of tuning requires significant effort and time from the security analyst.

My experience of manually labeling data indicates that the information that matters in determining the existence of attack tends to be highly specific to the nature of IDS alerts involved and contextual information within the network. In order for a machine-learning system to make high-quality classifications, such factors must be reflected in the feature selection. However, the challenge is how to encode fine-grained diverse information in a uniform format so that a machine-learning algorithm consumes. Current feature selection and construction as explained in Section 4.2.3 is coarse-grained, which may limit the effectiveness of the machine-learning approach.

Class Distribution

The class distribution (interesting vs. non-interesting) is skewed in this problem, with non-interesting scenarios significantly out-numbering interesting ones. This is consistent with past estimates of IDS false-positive alerts, where the estimation has been made that more than 99% of alerts reported by IDSs are not related to security issues[17]. This makes the classification process harder and prone to low accuracy[88]. In the labeled dataset, I found the ratio of positive "interesting" to negative "non-interesting" class to be approximately 1 : 4 because classification is based on whether a correlation is worthy of further investigation, opposed to whether a single event represents a true attack. Therefore, the latter would have a much lower ratio. This unbalanced data presents a challenge for machine learning because the resulting classifier could be biased towards the majority class. To ensure that both classes are learned with accepted performance, I experimented with balancing training data while maintaining original distribution in the testing data.

4.2.3 Feature Selection

The most important task in building a machine-learned classifier is to select features likely to be predictive to the classes of interest. Based on my empirical experience, I divided these features into two categories. The first category consists of information regarding input to SnIPS. Since SnIPS primarily takes Snort alerts as input evidence, I called this category "Snort-related features". The second category consists of information regarding SnIPS reasoning and correlation, manifested as structures of the correlation graphs. This category was called "SnIPS-related features". The feature set is described below.

Snort-Related Features

1. `Snort-signature set size.` In each correlation graph, a set of IDS alerts are triggered by a set of Snort signatures. This feature concerns the size of this set. For

example, in Figure 4.1, $sensor_1 - sensor_4$ represents Snort signatures, and the size of the signature set is 4.

2. `Snort-signature class groups.` In SnIPS, each Snort signature is given a weight (sensor quality metric). This weight is a measure of the trustworthiness level in alerts triggered by that signature. Snort signatures are categorized into 24 classes. *Class weight* is the maximum weight among all weights associated with Snort signatures in that class. This feature is represented as a k-size vector of pairs, where each pair consists of the number of appearance of a class and its corresponding weight, as follows:$\{\{\#\text{ of appearances of class\_type}_1\text{ , weight}_1\ \}\ , \cdots , \{\#\text{ of appearances of class\_type}_n\text{ , weight}_n\}\}$. If the class does not appear in the graph, I used zeros. For example, the graph in Figure 4.1 has $sensor_1$ and $sensor_2$ belong to $class_1$ and $sensor_3$ belongs to $class_2$. Therefore, this feature is $\{\{2, 0.33\}, \{1, 0.66\}, \{0, 0.0\}, \cdots , \{0, 0.0\}\}$, meaning that the first pair corresponds to $class_1$, which appears twice and has a weight of 0.33.

3. `Host categories.` I grouped monitored network IP addresses into three categories: client, server, and external IPs. This feature is a vector consisting of the number of appearances of the IPs in each category in the *Snort alerts* supporting the correlation. This feature is: $\{\{\#\text{ of appearances of client}\}, \{\#\text{ of appearances of server}\}, \{\#\text{ of appearances of external}\}\}$. For example, in Figure 4.1, suppose that ip_2 belongs to the server category, ip_3 and ip_4 belong to the client category, and ip_1 belongs to the external category. Thus, the IPs count of the $alert_1 - alert_5$ (not shown on the graph) is encoded as {4, 5, 1}.

SnIPS-Related Features

1. `Belief value of the correlation graph.` The maximum belief value of the sink nodes in a correlation graph is the belief value of the whole graph.

2. `Correlation-graph size.` This feature is the number of nodes in the graph. For

example, the correlation graph in Figure 4.1 has 9 nodes. Sensor nodes in dotted squares are not part of the graph.

3. `SnIPS' inference rules set.` This feature is pertains to the participation of SnIPS' inference rules that created the correlation graph. This feature is represented as a vector of SnIPS' inference rules appearing in the graph. Currently, SnIPS has three internal rules. Thus, this feature is: $\{\{\#$ of appearances of rule$_1\}$, $\{\#$ of appearances of rule$_2\}$, $\{\#$ of appearances of rule$_3\}\}$. If a rule does not appear in the graph, I assigned zero for it. For example, the sample graph in Figure 4.1 has this vector $\{1, 2, 0\}$ - $rule_1$ is used to connect Nodes 6 to 9, $rule_2$ is used to connect Nodes 7 to 9 and Nodes 8 to 9.

4.2.4 Learning Approaches

The nature of this problem implies that supervised or semi-supervised learning approaches are possible candidates. Supervised approaches often yield better results if enough labeled training data are available. Since the labeling process for this problem domain is time-consuming, it is oftentimes hard to have a large number of labeled samples. This allows for the possibility of applying semi-supervised learning techniques to address the scarcity of labeled data. By using semi-supervised learning, the system can start with a fraction of labeled data, and this classifier labels more data iteratively until the process stabilizes. Two approaches are well-known in semi-supervised learning: Co-training[24] and Expectation Maximization (EM)[36] and its variants, such as self-training (a.k.a., self-teaching or bootstrapping)[102]. In Section 4.3, results of using supervised and semi-supervised approaches are presented.

For various classification methods, Support Vector Machine (SVM) algorithm[33] has been used widely in the application of machine learning, including in cyber-security. SVM is a binary classifier in its original formulation. In the linearly separable case, it works by maximizing the separating boundary between the two classes, or margins, and selects a number of critical boundary instances, or support vectors, from each class. Then, it builds a linear discriminant function, or a hyperplane, separating the two classes. When the data is not linearly separable, the algorithm implicitly maps the data to a higher dimensional space, by means of a kernel, where data becomes linearly separable, and it builds a linear discriminant function in that space.

4.3 Implementation and Evaluation

In this section, I elaborate on implementation and experimentation of the machine learning approach.

4.3.1 Implementation

To facilitate the labeling process, I implemented a web-based interface that allows a security analyst to interact with the correlations-graphs list and determine whether the graph is interesting or non-interesting. Feedback is provided by two radio buttons (interesting and non-interesting) for each graph and recorded in a back-end database with the features extracted from the correlation graph and the supporting evidence (Snort alerts). I utilized $MySQL$, the open source database, for this purpose. The web interface was implemented in PHP and HTML, and the graphs visualization is generated using Graph Visualization Software (GraphViz)[41] tool. The graphs are displayed in the Scalable Vector Graphics (SVG) format, allowing the user to interact with the graph by issuing queries by clicking the nodes, thereby furthering analyzation of portions of correlation graph. In addition, to increase the ease of navigation, I implemented an additional interface method, which presents the graphs as a list of textual records. Each record represents a graph's sink node, *e.g.*, $(compromised(H1), timerange)$. Each compromised record can be clicked and checked for its supporting nodes. All experiment phase implementation was in Java using Weka[46] Java library to conduct experiments with supervised and semi-supervised.

4.3.2 Evaluation

In the experiments, I used a dataset with 1615 data points (Section 4.2.2). I conducted experiments with supervised and semi-supervised learning using SMO (Sequential Minimal Optimization) classifier, which is an implementation of the SVM algorithm from Weka[46].

Validation method

The best method to conduct machine-learning validation is to have multiple labeled datasets from different sources and experts. This is the ideal case but, in reality, I have one dataset labeled by one expert (me) and partially validated by another expert because labeling process is time consuming. This is discussed in more details in Section 4.2.2. Therefore, I used the n-fold cross-validation[101] method to evaluate results of my experiments. Even though, I utilized one dataset for training and testing, this method proved to have more validation power than percentage splitting the dataset into training and testing. In n-fold cross-validation, the original sample is randomly partitioned into *n* subsamples: *n-1* subsamples are used for training and the remaining one is used for testing. The procedure iterates *n* times to cover each possible split as testing exactly once. Empirically, 10 folds have been shown to give the most reliable results[101], so this method was used in my experiments.

Performance Metrics

I used accuracy, the area under the curve (AUROC), precision, recall, and F-measure to measure the performance. These measures ranged from (0 to 1), and the closer the measure to 1, the better it is. Even though accuracy is not a reliable measure in the case of skewed models, I chose to use it to observe how it behaves with the change of SMO's kernels and parameters. I also used the AUC because it has been shown to be a better classification performance measure compared to overall *accuracy*[26,50]. Receiver operating characteristic (ROC) Area Under Curve (AUC) is a way to measure classifier performance by obtaining different operating points in terms of true positive and false positive rates, and calculating the area under the curve using integral or some other approximation formula. Precision, recall, and F-Measure were used to better understand the classifier behavior for the two classes (interesting vs. non-interesting). Precision measures the fraction of classified instances relevant to the right class. Recall measures the fraction of correctly classified instances out of the total number of instances in that class. F-Measure is the harmonic mean of recall and precision measures, thereby showing the trade-off between precision and recall and offering a sense of how they perform.

$$\text{Accuracy} = \frac{\text{number of true positive} + \text{true negative}}{\text{number of true positive} + \text{false positive} + \text{true negative} + \text{false negative}}$$

$$\text{Precision} = \frac{\text{number of true positive}}{\text{number of true positive} + \text{false positive}}$$

$$\text{Recall (True Positive Rate)} = \frac{\text{number of true positive}}{\text{number of true positive} + \text{false negative}}$$

$$\text{False Positive Rate} = \frac{\text{number of false positive}}{\text{number of false positive} + \text{true negative}}$$

$$\text{F-measure} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

The following subsections describe the conducted experimentations, including the following:

1. Supervised experiments:
    - Support Vector Machine (SVM) kernels and parameters
    - Balance using over-sampling and under-sampling techniques
2. Semi-supervised experiments:

- Co-training
- Expectation Maximization (EM)

3. Benefit of machine learning approach experiments:
    - Comparison between customized DST prioritization (Chapter 3) and hybrid DST and machine learning.
    - Effort analysis to show how much effort is saved by this approach

Supervised Learning

Machine learning is a class of algorithms, which is data-driven. Supervised learning means that the examples must be fully labeled (in my case, 1615 labeled data points) so the algorithm can predict the classes. For various classification methods, SVM algorithm[33] has been used widely in the application of machine learning, including in cyber-security. SVM is a binary classifier in its original formulation. Therefore, I conducted all supervised experiments using Sequential Minimal Optimization (SMO) classifier, an implementation of the SVM algorithm from Weka[46]. The effectiveness of SVM depends on kernel selection, kernel's parameters, and soft margin parameter C. Therefore, I conducted the following experiments to select the best performance for SVM.

Support Vector Machine's Kernels Experiments I conducted experiments with multiple SMO kernels to find the best kernel for this problem. Choosing the right kernel function depends on the nature of the dataset, and the best mapping function is often determined experimentally in practice. This is done by applying various kernel functions and selecting the best kernel and parameters with the highest generalization performance on a validation dataset. I conducted experiments with normalized polynomial, polynomial, Gaussian Radial Basis Function (RBF) kernel, and Pearson VII Universal Kernel (PUK). PUK[97] is a universal kernel that can be calibrated to work as any standard SMO kernels by appropriately adjusting its two parameters, σ and ω. I used various parameter values provided in Üstün *et* *al.*'s[97] work to have the effect of standard kernels and other values for the parameters were used, too. Experiments indicated that use of the other values (called generic kernel) give the best performance metrics (Table 4.1). Therefore, I used these parameters to conduct the remaining experiments.

Soft Margin Parameter (C) Selection To avoid over fitting, SMO has a regularization parameter C that controls the trade-off between allowing training errors and enforcing rigid margins (a.k.a., bias-variance problem). Thus, this parameter can be seen as a penalty for training errors. Increasing the value of C increases the cost of misclassifying points and forces the creation of a classifier that fits the training dataset well but may not generalize well on new data. Decreasing the value of C may result in a very simple classifier that generally performs poorly. No widely accepted standard exists for selecting the appropriate value for C. One method recommends trying exponentially growing sequences, *e.g.*, $C \in \{2^{-5}, 2^{-3}, \cdots, 2^{13}, 2^{15}\}$. Another method uses $C \in \{0.5, 1, 2, 3, 5, 7, 10, 15, 20\}$[71].

Table 4.2 shows how the change in C parameter can affect results of the generic PUK kernel. In general, selecting the best parameter is is the responsibility of the user, because it depends on the nature of the dataset or network environment. I tried both and found that the best value for my experiments was 10.

Data Balancing As mentioned, the distribution of classes (interesting vs. non-interesting) is unbalanced, potentially resulting in models skewed toward the negative class (non-interesting), affecting the prediction ability for the positive case. As Table 4.1 shows, performance metrics for the interesting class are often worse than for the non-interesting class because the classifier does not have enough data to learn for prediction.

Several approaches address unbalance in classification. Over-sampling balances class populations through over-weighting minority class instances. For example, in this case, I used weight of 4 for each instance in the positive (interesting) class. Under-sampling balances class populations by eliminating majority instances[56]. In this application, I eliminate part of the negative (non-interesting) classes. Tables 4.3.2 and 4.3.2 illustrate results for over-sampling and under-sampling using SMO PUK generic kernel with parameter ($C = 10$).

Over-sampling results are shown in Table 4.3.2. As shown, the AUROC decreases in some cases and increases in others, but *Fmeasure* drops for all kernels. In addition, the recall for the interesting class increased and the precision decreased possibly because the simplicity in over-sampling method by artificially duplicating data samples.

Table 4.3.2 illustrates under-sampling results. Overall numbers dropped, meaning that under-sampling does not have good performance in this problem due to size and class-distribution ratio in the dataset. This is due to the small number of the instances the classifier learns after negative class trimming process. For the 1615 instances and 10-fold validation, I had 1449 instances for training and 161 for testing. Knowing that class distribution of the positive to negative is 1 : 4, means that the training set is approximately 575. This number is not enough for a classifier to generalize from. Therefore, I cannot rely on the under-sampling results.

Overall results in the over-sampling are better than under-sampling; however, there is no indication that over-sampling or under-sampling significantly improves classifier performance. This may be due to a number of factors such as the overall small number of labeled data and the simplicity in over-sampling by artificially duplicating data samples. In the following experiments, I used the unbalanced dataset.

Semi-Supervised Learning

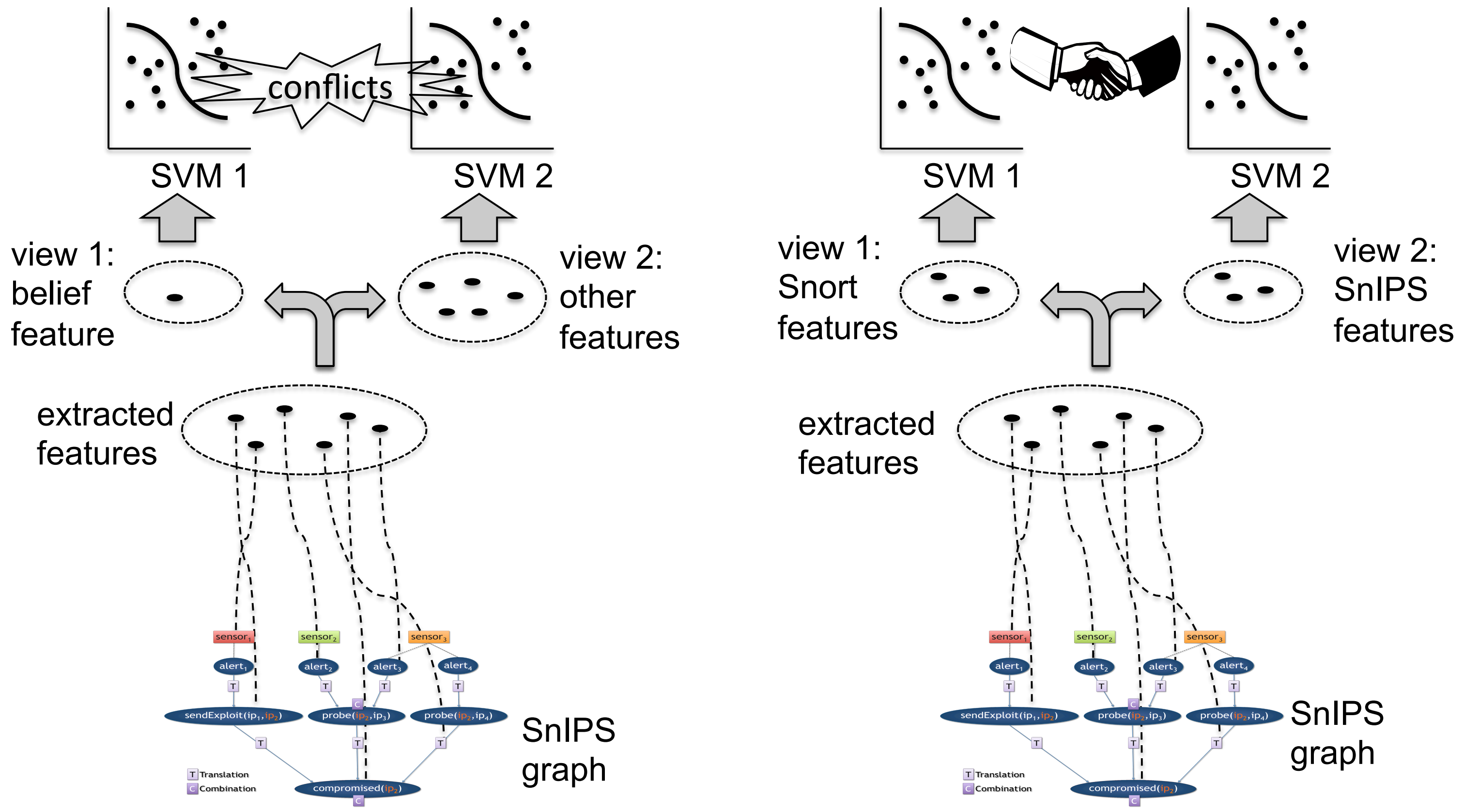
Semi-supervised learning means that the system can start with a very small amount of labeled data and grow by retraining itself until it consumes all unlabeled data. To validate my hypothesis (Section 4.1), I ran experiments with two widely used semi-supervised approaches: Co-training[24] and a popular variant of Expectation Maximization (EM)[36].

Co-training idea is to split the feature set into two subsets (views). Each view should be sufficient to build a classifier (predictive of the class label), and the two views should be conditionally independent given the class. The algorithm starts from a small-labeled sample and builds two classifiers from the two views. Then, it consumes some unlabeled instances from the unlabeled data pool, and each classifier labels them. Newly labeled data points are added to the training pool. Instances where the two classifiers disagree are ignored (thus, effectively removing possible noise in the labeling process). Next, the algorithm builds two new classifiers from the bigger labeled data pool, and iterates until the unlabeled data pool is consumed or a maximum number of iterations is reached[24,58,63]. I use SVM classifiers to learn from the two views in the co-training approach.

In the first experiment (co-training$_1$), I split the feature set into: the correlation graph belief value as the first view and the rest of the features as the second view. In the second experiment (co-training$_2$), I used SnIPS-related features for the first view and Snort-related features for the second view (section 4.2.3). This split is more intuitive and follows the two conditions for co-training.

Table 4.5 shows the comparison between the two approaches, where 10% of the training data is used as labeled and the rest of the data is used as unlabeled. For comparison, I also show an upper-bound, (the best results that SMO classifier can achieve with a fully labeled dataset, or a supervised learning) and a lower-bound (results for running SMO supervised classification only with 10% of labeled data from my dataset, as used in co-training). Co-training$_1$'s results are worse than the lower bound, which is certainly not acceptable because the two views were not conditionally independent since the belief value is effected by the other SnIPS' features. In addition, the belief value was not sufficient for prediction. Results show that co-training$_2$ seems to be the most promising approach. Figure 4.3illustrates a visual comparison between the two splits in co-training.

Table 4.5 also shows results for an Expectation Maximization (EM) type learning algorithm (self-training). EM-type learning is an iterative statistical technique for maximum likelihood estimation for small-labeled dataset. Given a model and data with missing class values, EM locally maximizes the likelihood of the parameters and predicts with estimates for missing class values[36]. At each iteration, instances from the unlabeled pool that are clas-



(a) Co-training$_1$ (b) Co-training$_2$

Figure 4.3: *Comparison between two ways of splitting features in co-training experiments*

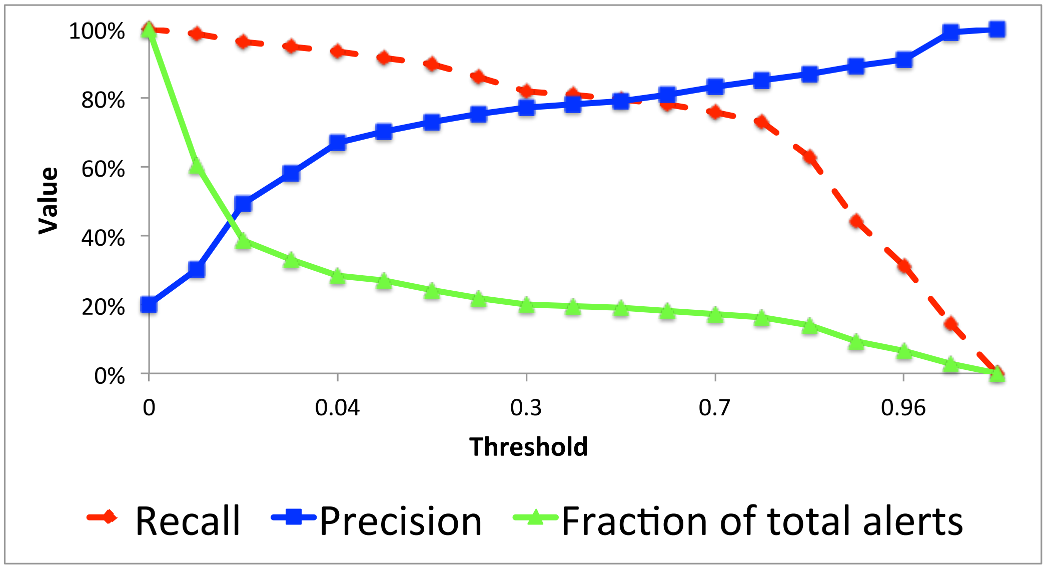


Figure 4.4: *Effort analysis using Support Vector Machine (SVM) (as supervised) learning, with σ=1,ω=1,and C=10*

sified with high confidence by the current classifier are added to the training data pool. As opposed to co-training, the use of one view makes it impossible to detect labeling conflicts. Self-training is also used with SMO in my experiments. Results show that this approach can also produce good performance but not as good as co-training because it experienced conflicts between some instances, such as noise, which self-training cannot detect. The two views used in co-training help in this respect since the two independent "experts" can point to conflicting instances.

Benefit of the Machine Learning Approach

Baseline Comparison I conducted an experiment to see if the machine learning approach is an improvement over the customized DST prioritization module. I used the belief value as a classifier criterion by thresholding the belief value of correlations and calculating performance metrics. Table 4.6 shows the advantage of the machine learning approach that uses *multiple* features, including the belief value, over the approach that relies *solely* on the customized DST belief value (denoted "Baseline" in the table).

Effort analysis I conducted an effort or benefit analysis using precision, recall, and dataset sample size to show the benefit of using machine learning to reduce the workload of the security analyst. I used logistic-models option in SMO to produce probability estimates for the output. Figure 4.4 shows the changes when the threshold (x-axis) increases. When the security analyst works a small amount of alerts correlations, the precision is high, meaning more effort is devoted to useful tasks, and the recall is high, too. When the workload increases, however, cumulative precision decreases as well. Analysis shows that the minimum amount of effort that gives good results can be achieved by choosing a 0.8 threshold that provides the security analyst, as little as 16% of the total correlations, 74% recall, and 83% precision. Furthermore, a sharp decline of the sample-size curve was observed, which can be measured by the vertical distance between the sample-size and precision curves. The precision curve starts from 20% on oppose of 0, because by choosing 0 as a threshold, the precision shows the amount of the true positive cases, which is 20% of the sample size. This is consistent with class distribution of the dataset, which is approximately 1 : 4. This result is a good indication that the classification helps presents interesting (potentially malicious) correlation to the security analyst.

Name	**(σ,ω)**	**Accuracy**	**Class***	**AUROC**	**Precision**	**Recall**	**F-Measure**
Generic	(1,1)	0.927	1	0.956	0.824	0.797	0.810
			0	0.956	0.951	0.958	0.955
			Weighted Avg.	0.956	0.926	0.927	0.926
POLY_1	(2.3,57)	0.919	1	0.953	0.797	0.785	0.791
			0	0.953	0.948	0.952	0.95
			Weighted Avg.	0.953	0.918	0.919	0.919
POLY_2	(1.9,90)	0.923	1	0.953	0.812	0.791	0.801
			0	0.953	0.950	0.955	0.952
			Weighted Avg.	0.953	0.923	0.923	0.923
RBF_1	(1.2,100)	0.922	1	0.957	0.809	0.791	0.800
			0	0.957	0.949	0.955	0.952
			Weighted Avg.	0.957	0.922	0.923	0.922
RBF_2	(4.7,85)	0.915	1	0.952	0.803	0.750	0.776
			0	0.952	0.940	0.955	0.948
			Weighted Avg.	0.952	0.913	0.915	0.914

* Class: interesting(positive) is "1" and non-interesting(negative) is "0".

Table 4.1: *Results for multiple SMO's PUK kernels values, with C=10, on unbalanced dataset*

Soft margin parameter (C)	Accuracy		Class*	AUROC	Precision	Recall	F-Measure
0.5	0.921		1	0.950	0.822	0.759	0.789
			0	0.950	0.943	0.960	0.951
		Weighted Avg.		0.950	0.919	0.921	0.920
1	0.922		1	0.952	0.830	0.756	0.791
			0	0.952	0.942	0.962	0.952
		Weighted Avg.		0.952	0.920	0.922	0.921
3	0.926		1	0.954	0.832	0.782	0.806
			0	0.954	0.948	0.962	0.955
		Weighted Avg.		0.954	0.925	0.926	0.925
5	0.927		1	0.952	0.830	0.788	0.808
			0	0.952	0.949	0.961	0.955
		Weighted Avg.		0.952	0.926	0.927	0.926
10	0.927		1	0.956	0.824	0.797	0.810
			0	0.956	0.951	0.958	0.955
		Weighted Avg.		0.956	0.929	0.929	0.929
20	0.928		1	0.955	0.821	0.813	0.817
			0	0.955	0.955	0.957	0.956
		Weighted Avg.		0.955	0.929	0.929	0.929
30	0.927		1	0.955	0.818	0.810	0.814
			0	0.955	0.954	0.956	0.955
		Weighted Avg.		0.955	0.927	0.928	0.927

* Class: interesting(positive) is “1” and non-interesting(negative) is “0”.

Table 4.2: *Results for multiple C values for the default SMO’s PUK kernel, with σ=1 and ω=1*

Name	(σ, ω)	Accuracy		Class*	ROC Area	Precision	Recall	F-Measure
Generic	(1,1)	0.916		1	0.958	0.752	0.843	0.795
				0	0.958	0.961	0.933	0.947
			Weighted Avg.		0.958	0.921	0.916	0.918
POLY_1	(2.3,57)	0.905		1	0.948	0.694	0.907	0.787
				0	0.948	0.976	0.904	0.939
			Weighted Avg.		0.948	0.921	0.905	0.909
POLY_2	(1.9,90)	0.907		1	0.948	0.701	0.907	0.791
				0	0.948	0.976	0.907	0.940
			Weighted Avg.		0.948	0.923	0.907	0.911
RBF_1	(1.2,100)	0.911		1	0.957	0.720	0.888	0.795
				0	0.957	0.972	0.917	0.944
			Weighted Avg.		0.957	0.923	0.922	0.915
RBF_2	(4.7,85)	0.905		1	0.946	0.696	0.901	0.786
				0	0.946	0.974	0.906	0.939
			Weighted Avg.		0.946	0.920	0.905	0.909

* Class: interesting(positive) is “1” and non-interesting(negative) is “0”.

Table 4.3: *Results for multiple SMO’s PUK kernels values with over-sampling balanced dataset, with C=10*

Name	(σ,ω)	Accuracy		Class*	ROC Area	Precision	Recall	F-Measure
Generic	(1,1)	0.892		1	0.953	0.659	0.924	0.769
				0	0.953	0.980	0.884	0.929
			Weighted Avg.		0.953	0.917	0.892	0.898
POLY_1	(2.3,57)	0.886		1	0.945	0.647	0.921	0.760
				0	0.945	0.979	0.878	0.925
			Weighted Avg.		0.945	0.914	0.886	0.893
POLY_2	(1.9,90)	0.877		1	0.945	0.628	0.908	0.743
				0	0.945	0.975	0.869	0.919
			Weighted Avg.		0.945	0.907	0.877	0.884
RBF_1	(1.2,100)	0.880		1	0.949	0.636	0.908	0.748
				0	0.949	0.975	0.874	0.922
			Weighted Avg.		0.949	0.909	0.880	0.888
RBF_2	(4.7,85)	0.880		1	0.947	0.638	0.899	0.746
				0	0.947	0.973	0.876	0.922
			Weighted Avg.		0.947	0.907	0.880	0.888

* Class: interesting(positive) is "1" and non-interesting(negative) is "0".

Table 4.4: *Results for multiple SMO's PUK kernels values with w under-sampling balanced dataset, with C=10*

Approach	**Accuracy**	**AUROC**	**Precision**	**Recall**	**F-Measure**
Lower bound	0.875	0.900	0.880	0.875	0.877
Co-training$_1$	0.833	0.807	0.819	0.833	0.797
Co-training$_2$	0.900	0.948	0.907	0.900	0.903
Expectation Maximization (EM)	0.893	0.895	0.889	0.893	0.890
Upper bound	0.927	0.956	0.926	0.927	0.926

Table 4.5: *Results for semi-supervised with SMO's generic PUK kernel on unbalanced data, with σ=1,ω=1,and C=10*

Approach	**Accuracy**	**AUROC**	**Precision**	**Recall**	**F-measure**
Baseline	0.803	0.679	0.755	0.803	0.753
Semi-supervised	0.900	0.948	0.907	0.900	0.903
Supervised	0.927	0.956	0.926	0.927	0.926

Table 4.6: *Comparison between using only SnIPS' belief value as a classifier (baseline), co-training$_2$ (semi-supervised) learning, and machine learning using SVM (supervised): with σ=1,ω=1,and C=10*

Chapter 5

Conclusion

In this dissertation, I present a method of handling uncertainty in intrusion analysis by combining multiple approaches. Uncertainty is created by an overwhelming amount of false alerts triggered by Intrusion Detection Systems (IDS). The source of this problem is deeply rooted in the base-rate fallacy phenomenon from which many detectors suffer. This phenomenon is revealed when a sensor probabilistically looks for an attack in large amount of benign traffic, leading to the mistaken declaration of benign traffic as the culprit (Appendix B). Handling uncertainty in intrusion analysis can help reducing the amount of false alerts security analyst must process each day. This can be achieved by causing intrusion alerts most likely to be true to stand out from the crowd. Therefore, I hypothesized that *the prioritization of IDS-correlated-alerts scenarios using a customized version of Dempster-Shafer Theory (DST) and machine learning can reduce uncertainty in intrusion analysis.* I used SnIPS[67] as a foundation for implementation and validation of the proposed methods. SnIPS is an intrusion analysis tool that builds dynamic abstract proof traces supported by evidences. I implemented a correlation engine that auto correlates SnIPS' proof traces to produce a complete attack scenario. This scenario is visualized as an inference graph through which the security analyst can navigate.

To prioritize correlation graphs, I needed a method to quantify uncertainty. I found that DST has proven its superiority in this problem (Chapter 3). Therefore, I propose a way to handle uncertainty in intrusion analysis using my customized version of DST. The proposed

DST customization can accurately combine non-independent evidence commonly found in correlated IDS alerts. In addition, I created a DST model for capturing sensor quality (Section 3.2.1), and an efficient algorithm for calculating belief values for the hypotheses in an alert correlation graph (Section 3.2.4). Evaluation was conducted for the approach on multiple datasets. Results of evaluation strongly indicate that the ranking provided by belief value gives good prioritization on correlated alerts based on their likelihood of being true attacks. I added my method implementation to SnIPS and have provided it as open source tool with the same name as SnIPS[54]. Furthermore, I deployed SnIPS in the CIS departmental network for continued evaluation.

Observation of the output of SnIPS running on the departmental network revealed that a security analyst must perform repetitive tasks in order to prune out false positives in correlation graphs. The reason behind this issue is inability to feedback the system with the security analyst's decisions. Therefore, I hypothesized that such repetitive tasks can yield (limited) labeled data, enabling the use of a machine learning-based approach to prune SnIPS output based on the human analysts' feedback, similar to spam filters that learn from users' past judgment to prune emails. I manually labeled SnIPS' output correlations and built prediction models using supervised and semi-supervised learning approaches. Experimentation results indicate that semi-supervised learning using Support Vector Machine (SVM) is promising approach for a practical machine learning-based tools that can aid security analysts when a *limited* amount of labeled data is available. Furthermore, the experiments revealed a number of interesting observations that give insights into the pitfalls and challenges of applying machine learning in intrusion analysis. The main lesson learned from this work is that proper labeling is an important step in this research and the insights in labeling helped in feature selection and construction. The research also indicated that encoding all relevant features to build a classifier is a challenging task and researcher must make trade-offs in deciding granularity of features.

The conclusion of this dissertation is that the prioritization of IDS correlated alerts

(Chapter 2) using the customized version of DST (Chapter 3) and semi-supervised learning technique from machine learning (Chapter 4), can help reduce uncertainty in intrusion analysis, which is shown by multiple experiments.

As future work, potential opportunity exists to apply the extended version of DST on host-domain access graph in order to produce predictions for benign and malicious domains, depending on a small set of labels. In addition, I plan to continue to explore the base-rate fallacy problem in order gain a better understanding of this problem, especially in other fields, which may help in significantly mitigating or eliminating its effect in intrusion analysis (Appendix B).

Bibliography

[1] Fake app attack: Fake av website 4. http://www.symantec.com/security_response/attacksignatures/detail.jsp?asid=23509.

[2] Federal rules of evidence. http://www.law.cornell.edu/rules/fre/.

[3] Mcafee trustedsource. http://www.trustedsource.org.

[4] The mebroot and tidserv crossover. http://www.symantec.com/connect/blogs/mebroot-and-tidserv-crossover.

[5] Norton safe web. http://safeweb.norton.com.

[6] System infected: Bredolab activity. http://www.symantec.com/security_response/attacksignatures/detail.jsp?asid=23363.

[7] System infected: Https tidserv request. http://www.symantec.com/security_response/attacksignatures/detail.jsp?asid=23570.

[8] System infected: Tidserv activity 2. http://www.symantec.com/security_response/attacksignatures/detail.jsp?asid=23615.

[9] Trend micro site safety center. http://global.sitesafety.trendmicro.com.

[10] Understanding evidence. http://www.lexisnexis.com/lawschool/study/outlines/pdf/evid.pdf.

[11] Web attack: Adobe acrobat suspicious executable download. http://www.symantec.com/security_response/attacksignatures/detail.jsp?asid=23218.

[12] Web attack: Malicious injected javascript 3. http://www.symantec.com/security_response/attacksignatures/detail.jsp?asid=23741.

[13] Web attack: Neosploit toolkit website 2. http://www.symantec.com/security_response/attacksignatures/detail.jsp?asid=23613.

[14] Worldwide intelligence network environment (wine). http://www.symantec.com/about/profile/universityresearch/sharing.jsp.

[15] Merriam-webster, 2014.

[16] Magnus Almgren, Ulf Lindqvist, and Erland Jonsson. A multi-sensor model to improve automated attack detection. In *11th International Symposium on Recent Advances in Intrusion Detection (RAID 2008)*. RAID, September 2008.

[17] Stefan Axelsson. The base-rate fallacy and its implications for the difficulty of intrusion detection. In *Proceedings of the 6th ACM Conference on Computer and Communications Security (CCS)*, 1999.

[18] Stefan Axelsson. The base-rate fallacy and the difficulty of intrusion detection. *ACM Trans. Inf. Syst. Secur.*, 3(3):186–205, 2000.

[19] Rebecca Bace and Peter Mell. Intrusion detection systems. Technical report.

[20] Rebecca Gurley Bace. *Intrusion Detection*. Sams Publishing, 2000.

[21] Maya Bar-Hillel. The base-rate fallacy in probability judgments. *Acta Psychologica*, 44(3):211 – 233, 1980.

[22] Marco Barreno, Alvaro A. Cárdenas, and J. D. Tygar. Optimal ROC curve for a combination of classifiers. In *Advances in Neural Information Processing Systems (NIPS)*, page 2008, 2007.

[23] Justin Beaver, Christopher Symons, and Robert Gillen. A learning system for discriminating variants of malicious network traffic. In *8th Cyber Security and Information Intelligence Research Workshop*, January 2013.

[24] Avrim Blum and Tom Mitchell. Combining labeled and unlabeled data with co-training. In *Proceedings of the 11th annual conference on Computational Learning Theory (COLT)*, 1998.

[25] Damiano Bolzoni, Sandro Etalle, and Pieter H. Hartel. Panacea: Automating attack classification for anomaly-based network intrusion detection systems. In *Proceedings of the 12th International Symposium on Recent Advances in Intrusion Detection (RAID)*, 2009.

[26] Andrew P. Bradley. The use of the area under the roc curve in the evaluation of machine learning algorithms. *Pattern Recognition*, 30, 1997.

[27] Qi Chen and Uwe Aickelin. Anomaly detection using the Dempster-Shafer method. In *International Conference on Data Mining (DMIN2006)*, 2006.

[28] Thomas M. Chen and Varadharajan Venkataramanan. Dempster-Shafer theory for intrusion detection in ad hoc networks. *IEEE Internet Computing*, 2005.

[29] Steven Cheung, Ulf Lindqvist, and Martin W Fong. Modeling multistep cyber attacks for scenario recognition. In *DARPA Information Survivability Conference and Exposition (DISCEX III)*, pages 284–292, Washington, D.C., 2003.

[30] Steven Cheung, Ulf Lindqvist, and Martin W Fongx. An online adaptive approach to alert correlationx. In *DARPA Information Survivability Conference and Exposition (DISCEX III)*, 2003.

[31] Chien-Yi Chiu, Yuh-Jye Lee, Chien-Chung Chang, Wen-Yang Luo, and Hsiu-Chuan Huang. Semi-supervised learning for false alarm reduction. In *Proceedings of the 10th industrial conference on Advances in data mining: applications and theoretical aspects(ICDM)*, 2010.

[32] Robert Cole. *Multi-step Attack Detection via Bayesian Modeling Under Model Parameter Uncertainty*. PhD thesis, The Pennsylvania State University, 2013.

[33] Corinna Cortes and Vladimir Vapnik. Support-vector networks. In *Machine Learning*, 1995.

[34] Frédéric Cuppens and Alexandre Miège. Alert correlation in a cooperative intrusion detection framework. In *IEEE Symposium on Security and Privacy*, 2002.

[35] H. Debar and A. Wespi. Aggregation and correlation of intrusion-detection alerts. In *Recent Advances in Intrusion Detection, LNCS*, 2001.

[36] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *journal of the royal statistical society, series b*, 39, 1977.

[37] Arthur P Dempster. The dempster–shafer calculus for statisticians. *International Journal of Approximate Reasoning*, 48(2):365–377, 2008.

[38] Thierry Denceux. The cautious rule of combination for belief functions and some extensions. In *9th International Conference on Information Fusion*, 2006.

[39] Dorothy Denning. An intrusion-detection model. *IEEE Transactions on Software Engineering*, 13(2), 1987.

[40] Cory Doctorow. *little Brother*. 2007.

[41] John Ellson, Emden Gansner, Lefteris Koutsofios, Stephen North, Gordon Woodhull, Short Description, and Lucent Technologies. Graphviz — open source graph drawing tools. In *Lecture Notes in Computer Science*, pages 483–484. Springer-Verlag, 2001.

[42] J. Goodman, D. Heckerman, and R. Rounthwaite. Stopping spam. *Scientific American*, 292(4), 2005.

[43] Nico Görnitz, Marius Kloft, Konrad Rieck, and Ulf Brefeld. Active learning for network intrusion detection. In *Proceedings of the 2nd ACM workshop on Security and artificial intelligence*, 2009.

[44] Paul Graham. *Hackers and Painters: Big Ideas from the Computer Age.* O'Reilly, 2004.

[45] Guofei Gu, Alvaro A. Cárdenas, and Wenke Lee. Principled reasoning and practical applications of alert fusion in intrusion detection systems. In *Proceedings of the 2008 ACM symposium on Information, computer and communications security*, ASIACCS '08, pages 136–147, New York, NY, USA, 2008. ACM.

[46] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The weka data mining software: an update. *SIGKDD Explor. Newsl.*, 11(1), 2009.

[47] Joseph Y. Halpern. *Reasoning about uncertainty.* The MIT Press, London,England, 2005.

[48] Steven Andrew Hofmeyr. *An Immunological Model of Distributed Detection and Its Application to Computer Security.* PhD thesis, University of New Mexico, 1999.

[49] Wenjie Hu, Yihua Liao, and V. Rao Vemuri. Robust anomaly detection using support vector machines. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2003.

[50] Jin Huang and Charles X. Ling. Using auc and accuracy in evaluating learning algorithms. *IEEE Transactions on Knowledge and Data Engineering*, 17, 2005.

[51] G. Hulten and J. Goodman. *Tutorial on junk e-mail filtering*, 2004.

[52] Paul Innella and Oba McMillan. An introduction to intrusion detection systems. http://www.securityfocus.com/infocus/1520, December 2001.

[53] Jonathan J. Koehler. The base rate fallacy reconsidered: Descriptive, normative, and methodological challenges. *Behavioral and Brain Sciences*, 19:1–53, 1996.

[54] Argus Lab. Snort intrusion analysis using proof strengthening (SnIPS). http://people.cis.ksu.edu/~xou/argus/software/snips.

[55] Pavel Laskov, Patrick Düssel, Christin Schäfer, and Konrad Rieck. Learning intrusion detection: Supervised or unsupervised? In *Image Analysis and Processing – ICIAP*. Springer Berlin / Heidelberg, 2005.

[56] Shoushan Li, Zhongqing Wang, Guodong Zhou, and Sophia Yat Mei Lee. Semi-supervised learning for imbalanced sentiment classification. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence(IJCAI)*, 2011.

[57] Hung-Jen Liao, Kuang-Yuan Tung, Chun-Hung Richard Lin, and Ying-Chih Lin. Intrusion detection system: A comprehensive review. *Journal of Network and Computer Applications*, 2012.

[58] Charles X. Ling, Jun Du, and Zhi-Hua Zhou. When does co-training work in real data? In *Proceedings of the 13th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining - PAKDD*, 2009.

[59] M.V. Mahoney and P.K. Chan. An analysis of the 1999 DARPA/Lincoln Laboratory evaluation data for network anomaly detection. In *Proceedings of the Sixth International Symposium on Recent Advances in Intrusion Detection (RAID)*, 2003.

[60] John McHugh. Testing intrusion detection systems: a critique of the 1998 and 1999 DARPA intrusion detection system evaluations as performed by Lincoln Laboratory. *ACM Trans. Inf. Syst. Secur. (TISSEC)*, 3(4):262–294, 2000.

[61] Gaspar Modelo-Howard, Saurabh Bagchi, and Guy Lebanon. Determining placement of intrusion detectors for a distributed application through bayesian network modeling.

In *11th International Symposium on Recent Advances in Intrusion Detection (RAID 2008)*. RAID, September 2008.

[62] Benjamin Morin, Ludovic Mé, Hervé Debar, and Mireille Ducassé. A logic-based model to support alert correlation in intrusion detection. *Information Fusion*, 10(4):285–299, 2009.

[63] Kamal Nigam and Rayid Ghani. Analyzing the effectiveness and applicability of co-training. In *Proceedings of the ninth international conference on Information and knowledge management - CIKM*, 2000.

[64] Peng Ning, Yun Cui, Douglas Reeves, and Dingbang Xu. Tools and techniques for analyzing intrusion alerts. In *ACM Transactions on Information and System Security*, May 2004.

[65] Peng Ning, Yun Cui, and Douglas S. Reeves. Constructing attack scenarios through correlation of intrusion alerts. In *Proceedings of the 9th ACM Conference on Computer and Communications Security (CCS 2002)*, pages 245–254, 2002.

[66] Steven Noel, Eric Robertson, and Sushil Jajodia. Correlating Intrusion Events and Building Attack Scenarios Through Attack Graph Distances. In *20th Annual Computer Security Applications Conference (ACSAC 2004)*, pages 350– 359, 2004.

[67] Xinming Ou, S. Raj Rajagopalan, and Sakthiyuvaraja Sakthivelmurugan. An empirical approach to modeling uncertainty in intrusion analysis. In *Annual Computer Security Applications Conference (ACSAC)*, 2009.

[68] Tadeusz Pietraszek. Using adaptive alert classification to reduce false positives in intrusion detection. In *Recent Advances in Intrusion Detection*, 2004.

[69] Tadeusz Pietraszek and Axel Tanner. Data mining and machine learning—towards reducing false positives in intrusion detection. 2005.

[70] Prasad Rao, Konstantinos F. Sagonas, Terrance Swift, David S. Warren, and Juliana Freire. XSB: A system for efficiently computing well-founded semantics. In *Proceedings of the 4th International Conference on Logic Programming and Non-Monotonic Reasoning (LPNMR'97)*, pages 2–17, Dagstuhl, Germany, July 1997. Springer Verlag.

[71] G. Rätsch, S. Sonnenburg, and B. Schölkopf. Rase: recognition of alternatively spliced exons in c.elegans. *Bioinformatics*, 21(1), 2005.

[72] H. Ren, N. Stakhanova, and A. Ghorbani. An online adaptive approach to alert correlationx. In *The Conference on Detection of Intrusions and Malware and Vulnerability Assessment (DIMVA)*, 2010.

[73] M.H. Rheinfurth and L.W. Howell. Probability and statistics in aerospace engineering. Technical report, Marshall Space Flight Center, Marshall Space Flight Center, 1998.

[74] Konrad Rieck. *Machine Learning for Application-Layer Intrusion Detection*. PhD thesis, Technische Universität, Berlin, 2009.

[75] Konrad Rieck. Self-learning network intrusion detection. *Information Technology IT*, 53(3), 2011.

[76] William Robertson, Federico Maggi, Christopher Kruegel, and Giovanni Vigna. Effective Anomaly Detection with Scarce Training Data. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, San Diego, CA USA, 02 2010.

[77] O. Rottenstreich and I. Keslassy. The bloom paradox: When not to use a bloom filter? In *INFOCOM, 2012 Proceedings IEEE*, 2012.

[78] L.J. Savage. *The Foundations of Statistics*. Dover Publications, 1972.

[79] Bruce Schneier. Biometrics in airports. http://www.schneier.com/crypto-gram-0109a.html, 2001.

[80] K. Sentz and S. Ferson. Combination of evidence in Dempster-Shafer theory. Technical report, Sandia National Laboratories, Albuquerque, New Mexico., 2002.

[81] G. Shafer. *A Mathematical Theory of Evidence*. Princeton University Press, 1976.

[82] G. Shafer. Probability judgement in artifical intelligence. In *Uncertainty in Artifical Intelligence*, 1986.

[83] Glenn Shafer. The problem of dependent evidence. Technical report, University of Kansas, 1984.

[84] Glenn Shafer. Belief functions and possibility measures. *The Analysis of Fuzzy Information*, 1986.

[85] Glenn Shafer. Probability judgment in artificial intelligence and expert systems. *Statistical Science*, 2(1), 1987.

[86] Chris Sinclair, Lyn Pierce, and Sara Matzner. An application of machine learning to network intrusion detection. In *Proceedings of the 15th Annual Computer Security Applications Conference*, ACSAC, 1999.

[87] R. Smith, N. Japkowicz, M. Dondo, and P. Mason. Using unsupervised learning for network alert correlation. *Advances in Artificial Intelligence*, pages 308–319, 2008.

[88] Robin Sommer and Vern Paxson. Outside the closed world: On using machine learning for network intrusion detection. In *IEEE Symposium on Security and Privacy (S&P)*, 2010.

[89] Sourcefire. Snort open source network intrusion prevention and detection system (ids/ips). http://www.snort.org.

[90] Lili Sun, Rajendra P. Srivastava, and Theodore J. Mock. An information systems security risk assessment model under Dempster-Shafer theory of belief functions. *Journal of Management Information Systems*, 22, 2006.

[91] Sathya Chandran Sundaramurthy, Loai Zomlot, and Xinming Ou. Practical IDS alert correlation in the face of dynamic threats. In *The International Conference on Security and Management (SAM)*, 2011.

[92] Helen Svensson and Audun Jϕsang. Correlation of intrusion alarms with subjective logic. In *sixth Nordic Workshop on Secure IT systems (NordSec)*, 2001.

[93] Christopher T. Symons and Justin M. Beaver. Nonparametric semi-supervised learning for network intrusion detection: combining performance improvements with realistic in-situ training. In *the 5TH ACM workshop on Artificial Intelligence and Security (AISec)*, 2012.

[94] Pang-Ning Tan, Sanjay Chawla, Chin Kuan Ho, and James Bailey, editors. *Advances in Knowledge Discovery and Data Mining - 16th Pacific-Asia Conference, PAKDD*, 2012.

[95] Yongning Tang and Ehab Al-Shaer. Sharing end-user negative symptoms for improving overlay network dependability. In *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2009.

[96] William C. Thompson and Edward L. Schumann. Interpretation of statistical evidence in criminal trials: The prosecutor's fallacy and the defense attorney's fallacy. *Law and Human Behavior*, 11(3):167–187, 1993.

[97] B. Üstün, W.J. Melssen, and L.M.C. Buydens. Facilitating the application of support vector regression by using a universal pearson vii function based kernel. *Chemometrics and Intelligent Laboratory Systems*, 81, 2006.

[98] H.L. Vacher. Quantitative literacy: Drug testing, cancer screening, and the identification of igneous rocks. *Journal of Geoscience Education*, 51:337–346, 2003.

[99] F. Valeur. *Real-Time Intrusion Detection Alert Correlation.* PhD thesis, University of California, Santa Barbara, May 2006.

[100] Fredrik Valeur, Giovanni Vigna, Christopher Kruegel, and Richard A. Kemmerer. A Comprehensive Approach to Intrusion Detection Alert Correlation. *IEEE Transactions on Dependable and Secure Computing*, 1(3):146–169, 2004.

[101] Ian H. Witten, Eibe Frank, and Mark A. Hall. *Data mining: practical machine learning tools and techniques.* Morgan Kaufmann Publishers Inc., 3rd edition.

[102] David Yarowsky. Unsupervised word sense disambiguation rivaling supervised methods. In *Proceedings of the 33rd annual meeting on Association for Computational Linguistics*, 1995.

[103] Dong Yu and Deborah Frincke. A novel framework for alert correlation and understanding. In *International Conference on Applied Cryptography and Network Security (ACNS)*, 2004.

[104] Dong Yu and Deborah Frincke. Alert confidence fusion in intrusion detection systems with extended Dempster-Shafer theory. In *43rd ACM Southeast Conference*, Kennesaw, GA, USA, 2005.

[105] Lotfi A. Zadeh. Book review: A mathematical theory of evidence. *AI Magazine*, 5(3):81–83, 1984.

[106] Y. Zhai, P. Ning, and J. Xu. Integrating IDS alert correlation and OS-level dependency tracking. *Intelligence and Security Informatics*, pages 272–284, 2006.

[107] Yan Zhai, Peng Ning, Purush Iyer, and Douglas S. Reeves. Reasoning about complementary intrusion evidence. In *Proceedings of 20th Annual Computer Security Applications Conference (ACSAC)*, pages 39–48, 2004.

[108] Zheng Zhang, Jun Li, C. N. Manikopoulos, Jay Jorgenson, and Jose Ucles. Hide: a hierarchical network intrusion detection system using statistical preprocessing and neural network classification. In *Proceedings IEEE Workshop on Information Assurance and Security*, 2001.

[109] J. Zhou, M. Heckman, B. Reynolds, A. Carlson, and M. Bishop. Modeling network intrusion detection alerts for correlation. *ACM Transactions on Information and System Security (TISSEC)*, 10(1):4, 2007.

[110] Loai Zomlot, Sathya Chandran, Doina Caragea, and Xinming Ou. Aiding intrusion analysis using machine learning. In *The* 12^{th} *International Conference on Machine Learning and Applications (ICMLA).*

[111] Loai Zomlot, Sathya Chandran Sundaramurthy, Kui Luo, Xinming Ou, and S.Raj Rajagopalan. Prioritizing intrusion analysis using DempsterShafer theory. In *the 4TH ACM workshop on Artificial Intelligence and Security (AISec)*, 2011.

Appendix A

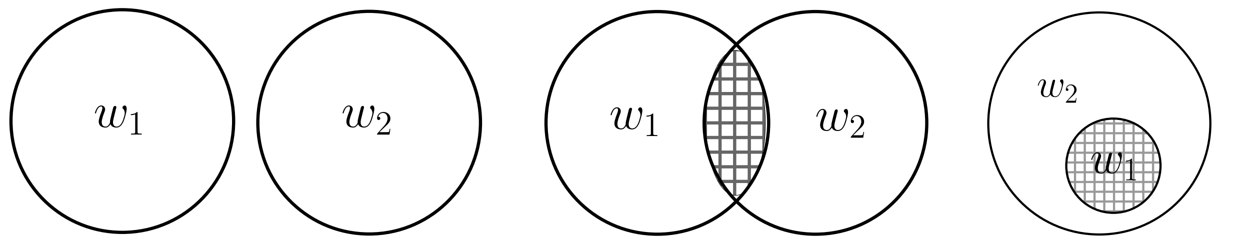
Mathematical Proof for Customized DST

Since only two non zero-bpa subsets, $true$ and θ, exist in each hypothesis's frame of discernment, w_i denotes the confidence in h_i ($h_i = t$), and $\overline{w_i}$ (negation of w_i) denotes the lack of confidence in h_i ($h_i = \theta$). The reader may think using w_i and $\overline{w_i}$, which appears to be not mutually exclusive, counterintuitive since θ includes $true$ and $false$. This is exactly how DST uniquely expresses disbelief in a hypothesis: It clearly differentiates between not believing a hypothesis and believing the negation of that hypothesis. When a person trusts a hypothesis, he/she believes its state is $true$, and when he/she does not trust a hypothesis, he/she is unsure of its state, hence θ. Interested readers are referred to Shafer's discussion on how to handle non-independent evidence using this interpretation[85].

The semantics of overlapping factor can be defined as:

$$r_1 = \frac{Pr[w_2|w_1] - Pr[w_2]}{Pr[\overline{w_2}]}, \quad r_2 = \frac{Pr[w_1|w_2] - Pr[w_1]}{Pr[\overline{w_1}]}$$

Without loss of generality, let us take r_1 as an example to explain the semantics. If we condition on trusting hypothesis h_1, the probability that we also trust h_2 is greater than or equal to its absolute probability since shared IDS sensors give the user only positive correlation. The bigger the difference, the stronger influence trusting h_1 has on trusting h_2. The extreme case is when $Pr[w_2|w_1] = 1$, which gives $r_1 = 1$. Another extreme case when



(a) Independent (b) Non-independent (c) Completely dependent

Figure A.1: *Venn diagram illustration of evidence dependency*

they are independent, thus $r_1 = 0$ as shown here:

by using bayes' rule,

$$r_1 = \frac{Pr[w_1, w_2] - Pr[w_1] \cdot Pr[w_2]}{Pr[\overline{w_2}] \cdot Pr[w_1]}$$

when w_1 and w_2 are independent then,

$$r_1 = \frac{Pr[w_1] \cdot Pr[w_2] - Pr[w_1] \cdot Pr[w_2]}{Pr[\overline{w_2}] \cdot Pr[w_1]}$$
$$r_1 = 0$$

when w_1 and w_2 are totally dependent (Figure A.1) then,

$$r_1 = \frac{1 - Pr[w_2]}{Pr[\overline{w_2}]} = \frac{Pr[\overline{w_2}]}{Pr[\overline{w_2}]},$$
$$r_1 = 1$$

Both r_1 and r_2 measure the dependence between w_1 and w_2, but from different directions.

Theorem A.0.1.

$$r_2 = \alpha \cdot r_1, \ \textit{where}\ \alpha = \frac{Pr[w_1] \cdot Pr[\overline{w_2}]}{Pr[w_2] \cdot Pr[\overline{w_1}]} \tag{A.1}$$

Proof.

$$r_1 \cdot Pr[\overline{w_2}] \cdot Pr[w_1] = Pr[w_1, w_2] - Pr[w_1] \cdot Pr[w_2]$$
$$r_2 \cdot Pr[\overline{w_1}] \cdot Pr[w_2] = Pr[w_1, w_2] - Pr[w_1] \cdot Pr[w_2]$$

We then have,

$$r_1 \cdot Pr[\overline{w_2}] \cdot Pr[w_1] = r_2 \cdot Pr[\overline{w_1}] \cdot Pr[w_2]$$

knowing that $\overline{w_i} = 1 - w_i$, then:

$$r_2 = r_1 \cdot \frac{Pr[w_1] \cdot Pr[\overline{w_2}]}{Pr[w_2] \cdot Pr[\overline{w_1}]}$$

□

Considering the following customized DST formulas,

$$\psi[t, t] = r_1 \cdot m_1(t) + (1 - r_1) \cdot m_1(t) \cdot m_2(t)$$
$$\psi[t, \theta] = (1 - r_1) \cdot m_1(t) \cdot m_2(\theta)$$
$$\psi[\theta, t] = (1 - r_2) \cdot m_1(\theta) \cdot m_2(t)$$
$$\psi[\theta, \theta] = r_1 \cdot m_2(\theta) + (1 - r_1) \cdot m_1(\theta) \cdot m_2(\theta)$$

and substitute the following definitions:

$$m_i(t) = Pr[w_i] \quad m_i(\theta) = Pr[\overline{w_i}]$$

Theorem A.0.2. *Based on the above definitions*

$$\psi[h_1, h_2] = Pr[w_1, w_2]$$

Proof. This proof is split into four small sub-proofs for each formula.

part (a).

To prove $Pr[w_1, w_2] = \psi[t, t]$, let us do the following:

$$Pr[w_2|w_1] = \frac{Pr[w_1, w_2]}{Pr[w_1]}$$

substitute the above into the definition of r_1, we get

$$r_1 \cdot Pr[\overline{w_2}] \cdot Pr[w_1] = Pr[w_1, w_2] - Pr[w_1] \cdot Pr[w_2]$$

knowing that $Pr[\overline{w_2}] = 1 - Pr[w_2]$, then:

$$\begin{aligned} &Pr[w_1, w_2] = r_1 \cdot Pr[w_1] + (1 - r_1) \cdot Pr[w_1] \cdot Pr[w_2] \\ &= \psi[t, t] \end{aligned}$$

□

part (b).

To prove $Pr[w_1, \overline{w_2}] = \psi[t, \theta]$, let us do the following:

$$\begin{aligned} &Pr[w_2|w_1] = 1 - Pr[\overline{w_2}|w_1] \\ &Pr[\overline{w_2}|w_1] = \frac{Pr[w_1, \overline{w_2}]}{Pr[w_1]} \text{ then,} \\ &Pr[w_2|w_1] = \frac{Pr[w_1] - Pr[w_1, \overline{w_2}]}{Pr[w_1]} \end{aligned}$$

substitute the above into the definition of r_1, we get

$$r_1 \cdot Pr[\overline{w_2}] \cdot Pr[w_1] = Pr[w_1] - Pr[w_1, \overline{w_2}] - Pr[w_1] \cdot Pr[w_2]$$

knowing that $Pr[\overline{w_1}] = 1 - Pr[w_1]$, then:

$$\begin{aligned} &Pr[w_1, \overline{w_2}] = (1 - r_1) \cdot Pr[w_1] \cdot Pr[\overline{w_2}] \\ &= \psi[t, \theta] \end{aligned}$$

□

part (c).

To prove $Pr[\overline{w_1}, w_2] = \psi[\theta, t]$, let us do the following:

$$Pr[w_1|w_2] = 1 - Pr[\overline{w_1}|w_2]$$
$$Pr[\overline{w_1}|w_2] = \frac{Pr[\overline{w_1}, w_2]}{Pr[w_2]} \text{ then,}$$
$$Pr[w_1|w_2] = \frac{Pr[w_2] - Pr[\overline{w_1}, w_2]}{Pr[w_2]}$$

substitute the above into the definition of r_2, we get

$$r_2 \cdot Pr[\overline{w_1}] \cdot Pr[w_2] = Pr[w_2] - Pr[\overline{w_1}, w_2] - Pr[w_1] \cdot Pr[w_2]$$

knowing that $Pr[\overline{w_1}] = 1 - Pr[w_1]$, then:

$$Pr[\overline{w_1}, w_2] = (1 - r_2) \cdot Pr[\overline{w_1}] \cdot Pr[w_2]$$
$$= \psi[\theta, t]$$

$\square$

part (d).

To prove $Pr[\overline{w_1}, \overline{w_2}] = \psi[\theta, \theta]$, let us do the following:

by using r_1 definition then,

$1 - r_1 = \dfrac{Pr[\overline{w_2}|w_1]}{Pr[\overline{w_2}]}$

from above two definitions of r_1 then,

$(1 - r_1) \cdot Pr[\overline{w_1}] \cdot Pr[\overline{w_2}] = Pr[\overline{w_2}|w_1] \cdot Pr[\overline{w_1}]$

$r_1 \cdot Pr[\overline{w_2}] = Pr[w_2|w_1] - Pr[w_2]$

by adding the two equitions we have,

$(1 - r_1) \cdot Pr[\overline{w_1}] \cdot Pr[\overline{w_2}] + r_1 \cdot Pr[\overline{w_2}] = Pr[\overline{w_2}|w_1] \cdot Pr[\overline{w_1}] + Pr[\overline{w_2}|w_1] - Pr[w_2]$

by substituting from the above Dempster's extended fromulas,

and knowing $Pr[\overline{w_1}] = 1 - Pr[w_1]$ and $Pr[\overline{w_2}, w_1] = Pr[\overline{w_2}|w_1] \cdot Pr[w_1]$ we have,

$Pr[\overline{w_1}, \overline{w_2}] = Pr[\overline{w_2}] - Pr[w_1, \overline{w_2}]$

by substituting for $Pr[w_1, \overline{w_2}]$ from above formulas and knowing $Pr[w_1] = 1 - Pr[\overline{w_1}]$, we have,

$Pr[\overline{w_1}, \overline{w_2}] = r_1 \cdot Pr[\overline{w_2}] + (1 - r_1) \cdot Pr[\overline{w_1}] \cdot Pr[\overline{w_2}]$

$= \psi[\theta, \theta]$

$\square$

$\square$

The importance of this theorem is that my calculation of the joint *bpa* $\psi[h_1, h_2]$ is sound because it gives a generalization of the joint probability distribution of the trustworthiness of two non-independent sources. This also follows Shafer's general guide on how to handle non-independent evidence sources in DST[85], although Shafer did not provide specific formulations.

Appendix B

Base-rate Fallacy in Intrusion Analysis

The base rate fallacy is a corollary of the Bayes Theorem that links a conditional probability of an event to that of its negation. It can be explained using the following example.

Example B.0.1. *Suppose a medical test is* 99% *accurate, meaning that when the test is administered to* 100 *people all of whom have the disease,* 99 *of the test results will be positive; when the test is administrated to* 100 *people none of whom has the disease,* 99 *of the test results will be negative. Suppose that your doctor has asked you to take this test, and the result came back positive. Does this mean that you need to assume that you have the disease? The answer to this question depends very strongly on the prevalence of the disease for the overall population. Suppose that the disease is very rare, say with* $1 : 1,000,000$ *prior probability (only 1 in a million people has this disease). After knowing this, one can say with high confidence that the test result is most likely wrong. This is due to the following calculation.*

$$\begin{aligned} Pr[D|T] &= \frac{Pr[T|D] \cdot Pr[D]}{Pr[T|D] \cdot Pr[D] + Pr[T|\neg D] \cdot Pr[\neg D]} \\ &= \frac{0.99 \cdot 0.000001}{0.99 \cdot 0.000001 + 0.01 \cdot 0.999999} \\ &\approx 0.01\% \end{aligned}$$

where T: test positive and D: has the disease.

This calculation shows that even when the test result is positive, it can be ignored since only 0.01% chance exists that you have this disease. Therefore, false positive tests are far more probable than true positive tests when the overall population has a low incidence of the disease. This phenomenon is called the false-positive paradox; and neglecting this paradox is the cause of the base-rate fallacy. The primary cause of the above result is that false-positive rate for the administrated test $Pr[T|\neg D] = 0.01$. On the other hand, the base-rate (prior) probability for a disease happens to be $Pr[D] = 0.000001$, Therefore, $Pr[T|\neg D]$ is *much greater than* $Pr[D]$, meaning that if the false-positive rate for the test is greater than the prior for the attack, detection is difficult using that *test*[73]. Therefore, when a person tries to detect an extremely weak signal, the test's accuracy must match the weakness of that signal. For example, if you are trying to point at a single pixel on your screen, a sharp pencil is a good pointer because the pencil-tip is smaller than a pixel. On the other hand, a pencil-tip is not suitable to point at a single atom in your screen[40].

Axelsson's Reasoning

Axelsson applied this base-rate reasoning to the intrusion detection problem in his 1999 paper[17] (and a subsequent journal version[18]). Let I denotes the event that an intrusion occurs and A denotes the event that the IDS sensor fires. The essence of the argument is that in order to measure the effectiveness of an IDS sensor, examination of true positive ($Pr[A|I]$) and false positive ($Pr[A|\neg I]$) is not sufficient. Bayesian detection rate ($Pr[I|A]$), likelihood of an attack when an alert is fired, need to be considered. Due to Bayes theorem, the latter is also dependent upon the prior probability that an intrusion will happen (the base rate $Pr[I]$). Based on reasonable assumptions about network operations at that time, Axelsson assumed that the probability an audit record will contain attack activities to be on the order of 10^{-5}. Formula below highlighting the relationship among Bayesian-detection rate, true-positive rate, and false-positive rate of an IDS sensor, can be obtained using similar calculation as in Example B.0.1.

$$\begin{aligned} Pr[I|A] &= \frac{Pr[A|I] \cdot Pr[I]}{Pr[A|I] \cdot Pr[I] + Pr[A|\neg I] \cdot Pr[\neg I]} \\ &= \frac{Pr[A|I] \cdot 10^{-5}}{Pr[A|I] \cdot 10^{-5} + Pr[A|\neg I] \cdot 0.99999} \end{aligned}$$

where A: IDS fires and I: intrusion occurs

The formula clearly indicates that in order to reach a reasonable Bayesian detection rate, approximately 50%, false-positive rate $Pr[A|\neg I]$ must be "on par" with base rate of intrusion events, which is approximately 10^{-5}, according to Axelsson. To build an IDS sensor with that low of a false positive while maintaining a reasonable true positive (*i.e.*, $Pr[A|I]$ close to 1), is infeasible in practice.

Axelsson's estimate of attack base rate is in accordance with the understanding of production networks and number of attacks that possibly seen therein at the time the paper was written. Currently, the number of attacks (or attack attempts) has increased dramatically because of the virtually non-existence of deterrence for malicious activities in cyber space. On the other hand, network bandwidth and CPU powers have also been increasing dramatically, leading to an even larger amount of legitimate traffic/events. Based on my experience of intrusion detection systems, the base rate for attacks is currently much lower than 10^{-5} [1] meaning that Axelsson's base-rate problem still exists.

Base-rate Fallacy in Other Fields

So far I have considered the problem of false-positive paradox (base-rate fallacy) in intrusion analysis first pointed out by Axelsson[18], which has made it virtually impossible to accurately detect intrusion by a *single sensor*. A very low false-positive rate will result in so many false alarms as to make the analysis useless in practice. However, this is a general phenomenon and the consideration of how other fields handle this problem would be useful.

[1]I ran the Snort IDS on a production network with the default rule set and it generated approximately 500,000 alerts daily, but intrusions are rarely found on the network.

In machine learning, base-rate phenomenon is revealed as highly imbalanced classifiers. For example, when classifying fraudulent actions in the banking systems, the number of fraudulent actions is much smaller than normal transactions in credit card usage data. In the case of two-class classifiers, this leads to submergence of the entire minority (positive) class into the majority (negative) class because the way classifier algorithm works by consistently aims to reduce error rate. There are some techniques,used in machine learning, to avoid this problem. In general, these techniques focus on increasing the base-rate of the minority class in the dataset before applying the primary classification algorithm. In concrete terms, this means one needs to increase the number of the samples for the minority class in the training phase. Moreover, it has been proven that the error in the highly skewed models is not related to the ratio but to the amount of samples of the class[94]. Therefore, if one has enough samples from the positive case (rare class), and a good classifier algorithm, with the right feature set, the algorithm will learn and will detect the minority class. However, for intrusion detection one does not have control over the occurrence of the rare class (malicious events) to increase their occurrence even during training phases (unless one is willing to deal with potentially non-representative synthetic data).

In the medical field, this phenomenon is well known, too. The problem is usually overcome by a multi-stage process. The first step is to screen with a test or procedure that yields low false negatives with an acceptable rate of true positives. Therefore, doctor will then have a *concentrated* high risk population. Secondly, the doctor diagnoses with a more complex procedure with low false-positive rate. For example, if breast cancer is regarded as a rare disease in a certain age group, women are recommended to do screening tests before any symptoms development. Then only if something *suspicious* is found during a screening exam, the doctor use one or more methods (usually costly) to find out if a disease is present. If cancer is found, other tests will be done to determine the stage (extent) of the cancer. This multi-step process helps in avoiding base-rate fallacy and reducing the total cost. Such a multi-step process is similar to how security practitioners seem to approach intrusion detection; however a systematic study of such process from mathematical probability of detection, is yet to be conducted.

In jurisprudence, the law of evidence is used to prove facts in a legal proceeding. The law of evidence considers amount, quality, and type of proof (evidence) needed to trial. There are rules to accept and reject evidences and even weigh an evidence. In a trial several types of evidence are required, *e.g.* , oral statements, physical objects, and documentary material. Persuading a court about a case, often requires multiple trustworthy-evidence pieces. For example, in Scottish law a rule requires two pieces of evidence to prove each essential fact, *e.g.*, DNA evidence could be corroborated with eyewitness testimony. Therefore, intuitively collaborating multiple relevant high quality sensors helps the court in decision process[2,10]. However, there is an interesting phenomenon in law, which is Prosecutor's Fallacy[96]; a known mistake in trial when prior probability of a defendant being guilty or innocent while considering an evidence is neglected.

Base-rate fallacy is also revealed in "bloom filter paradox" in computer science. Bloom filters are probabilistic data structures that can answer set membership queries with zero false negatives and very low probability of false positive, *e.g.*, $\leq 10^{-3}$. Networking devices typically use bloom filters as cache directories. Neglecting the prior set-membership probability of elements can actually make the directory more harmful than beneficial[77].

According to Schneier[79] base-rate fallacy can render useless security-face-recognition systems. Finally, base-rate fallacy is a phenomenon in many other fields such as Geosciences[98], Psychology, Behavioral, and Brain Sciences[21,53].